ARDUINO PROGRAMMING

A STEP BY STEP GUIDE TO LEARN ARDUINO PROGRAMMING FOR ABSOLUTE BEGINNERS

LILLY TRINITY

ARDUINO PROGRAMMING

A STEP BY STEP GUIDE TO LEARN ARDUINO PROGRAMMING FOR ABSOLUTE BEGINNERS

LILLY TRINITY

Arduino Programming

A Step by Step Guide to Learn Arduino Programming For Absolute Beginners

Lilly Trinity

Table of Content

Introduction

Thanks for downloading this ebook,

“Arduino Programming: A Step by Step Guide to Learn Arduino Programming For Absolute Beginners”

This book contains demonstrated moves and techniques on the best way to utilize Arduino in your tech projects. Arduino turned into a well-known explication that stretches out computing and applies autonomy to outside the innovation field. Dabblers can do these projects at home while increasing every one of the benefits of interest this product offers. This book will explain all concerning Arduino and the serviceable parts behind its functions. As a trainee, this book demonstrates you of the ideas, significant Arduino parts, essential coding basics, and some more. Towards the finish of the book, you 'll discover a few hints and skills, just as fledgling level project thoughts that will enable you to master Arduino!Thanks again for downloading this book.

I believe you appreciate it!

Chapter 1

What is Arduino?

The fantastic world of computing kept on stirring the minds of individuals interested in this field. They want to get their hands into technological projects using a simple circuit board and program codes. Arduino makes it possible for people outside the technology field to create their own devices with specific functions. In this section, you 'll learn about these points:

Arduino and its definition

– Where it ’ s used

– Available Arduino types

– Arduino ’ s limitations

Definition:

Arduino is a microcontroller created as an open-source framework. A chip controlled it and made out of various segments bound on the board. It looks like a mini motherboard utilized in a variety of projects. Arduino is additionally programmable as per the required functions in a project. Plans will execute to allowing individual pins to perform specific tasks. Parts and fasteners are classified using the names imprinted on the board. Usually, we know the term " Arduino " as the real smaller than a standard board. Nevertheless, the Arduino board needs to utilize its software version, additionally called Arduino software. It ' s used for programming commands that indicates the advisory group' s reason or capacity.

The Advantage of Using Arduino

Several individuals enjoyed this item as it is intended to make robotics and mini computing figuring available to regular clients. Arduino promotes for prototyping specialists, tenderfoot architects, and the individuals who need to attempt essential robotics regardless of the absence of engineering expertise. Everybody who needs to explore robotics and computing would now be able to do projects directly at their homes.

Another benefit is it's low-priced. An Andruino board ' s value begins at $20 and up subject to upon the number of installed parts, part types, and spaces. The price alone is reasonable for learners who are examining Arduino-controlled robotics and computing. Specialists can finish little activities, which don’ t usually cost a great deal of cash, yet at the same time offers the features required by developers.

Arduino ' s open-source, and programmable stage brings another advantage. Being a the open-source framework, Arduino can perform functions required by designers by transferring source codes to get their projects moving.

The long haul favorable position is utilizing Arduino can enable specialists to manufacture their very own boards. Clients learn Arduino's engineering by using the board and their capacities. Designers would then be able to customize their future boards as indicated by their projects complex system.

At long last, Arduino works with various parts, enabling designers to be progressively fun loving with their project ideas. Undertakings can be as necessary as actuating squinting LEDs or flickering or extends that are more mechanical.

What Projects can You Do with Arduino?

Arduino is a perfect gadget that gives developers a chance to do any project. Plain and straightforward projects include producing up a little computer for vehicles, web-based life " like " counters, MIDI controllers, and significantly more.

Limitations

Even though this framework enables hobbyists to do nearly everything, Arduino still has its restrictions. Its weakness to catch and record videos is its major drawback. The board's specifications are lacking to help these errands, which is altogether different from regular PCs and portable devices. These gadgets are intended for media recording and planned with proper components.

In any case, Arduino is capable of projecting pictures or designs through an outside presentation. Unlike capturing videos, anticipating won't use as many assets and capacity from the board. Additionally, using an exterior display will deal with information change to show pictures or other data. Developers must make a unique arrangement to make this setup feasible.

Available Types

Arduino comes in various models and types. Each model has one of a kind features and matches a particular function. From this moment on, Arduino can distribute in three models. Specific models are accessible in a few variations that take into account unique projects specifications.

Significant Things to Remember

Arduino has various PINs and parts relying upon the model. Misunderstanding the model will result in hatred. A few PINs may not work correctly when utilized on different boards.

Another issue is doing the wrong board can be mistaken for the developer. Project guides determine PINs and parts. Being a tenderfoot Arduino client, you may get muddled when you don't discover jumpers, PINs, and other indispensable parts for the project.

Keep away incompatibility issues by perusing the guide well. Confirm the necessary board before shopping. A few aides give a connection to the demonstrated Arduino model, which you can also click, buy the recommended board.

Arduino Development isn't Limited to Hardware Knowledge

Applying Arduino for a venture isn't constrained to understanding its parts and their particular functions. Your project’s prosperity also depends if the code is appropriately written and effectively stacked to the system.

Arduino requires learning the coding procedure and its underlying ideas. You should also realize how to work the product and designing codes.

Chapter 2

The advantages of Arduino

Most "micro-controller systems" are constrained to Windows as it were. Simple to learn for newcomers: The programming background is anything but difficult-to-use for learners, yet adaptable enough for cutting edge clients to exploit too. Another huge bit of advantage of Arduino is its library of precedents current inside the product of Arduino IDE.

1. **Low-priced**

Arduino boards are generally inexpensive contrasted with other microcontroller platforms. The most economical form of the Arduino module can rapidly compile by hand, and even the pre-assembled Arduino modules cost under $20.

2. **Ready-to-use**

The enormous favorable position of Arduino is it's ready to use formation. As Arduino arrives in a complete package which incorporates the 5V regulator, a burner, an oscillator, a micro-controller, sequential correspondence interface, LED and headers for the connections. You don't need to consider about programmer connections for programming or some other interface. Hooked it into the USB port of your PC, and that is it. Your progressive thought is going to change the world after only a couple of expressions of coding.

3. **Cross-platform**

The Arduino programming keeps running on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

4. **Simple functions**

During coding of Arduino, you will see a few features which make life so natural. Another benefit of Arduino is its automatic unit conversion ability. During the debugging process, you don't need to stress over the units changes. Only use everything power on the first pieces of your projects. You don't need to bother over side issues.

5. **A straightforward, programming environment**

The Arduino programming atmosphere is anything but difficult-to-use for newcomers, yet resilient enough for cutting edge clients to exploit too. For instructors, you can locate it on the Processing programming environment, so learners figuring out how to program in that environment will be intimate with the look and feel of Arduino.

6. **Open source and extensible programming**

The Arduino programming is available as open source apparatuses, accessible for expansion by experienced software engineers. The language is extensible through C++ libraries, and people need to understand the specialized subtleties can make the jump from Arduino to the AVR C programming language on which it is standing. So also, you can include AVR-C code straightforwardly into your Arduino programs if you need to.

7. **Open source and extensible equipment**

The Arduino depends on Atmel's ***ATMEGA8*** and ***ATMEGA168*** microcontrollers. The designs for the modules can distribute under a Creative Commons license, so experienced circuit designers can make their variant of the sharp edge, broadening it and improving it. Indeed, even generally, unpracticed clients can fabricate the breadboard rendition of the module to see how it functions and set aside money.

8. **Examples of codes**

Another preferred critical position of Arduino is its library of models exhibits inside the product of Arduino. I'll clarify this favorable position utilizing a model of voltage measurement. For instance, if you want to measure voltage using ATmega8 miniaturized scale controller and need to show the yield on a PC screen, at that point, you need to experience the entire procedure. The procedure will begin from learning the ADC's of smaller scale controller for measurement, experienced the learning of subsequent correspondence for presentation and will finish at USB - Serial converters. If you need to check this entire procedure.

9. **Enormous people group**

There are numerous gatherings present on the web in which individuals are discussing Arduino. Designers, specialists, and experts are doing their undertakings through Arduino. Deprived of many stretches, discover help with all the fixings. Besides, the Arduino site itself clarifies each capacity of Arduino.

Along these lines, We ought to finish up the benefit of Arduino by saying that during dealing with various undertakings, you need to stress over your creative thought. The rest of the will deal with by Arduino itself.

Chapter 3

Critical Terms in Arduino & Understanding the Choices

Along these lines, you chose to proceed to get yourself an Arduino; however, once it arrived, you understood you have no clue how to manage it.

Try not to freeze, for assistance is within reach! In this how-to, we will see how, to begin with, Arduino microcontroller sheets. We'll cover software installation, just as connecting and configuring the Arduino IDE.

You Will Need

- Arduino Uno Board (https://www.digikey.com/product-detail/en/arduino/A000073/1050-1041-ND/3476357)
- USB B Cable (most printers use this) OR USB Micro - B Cable (most Android phones or digital cameras use this)
- Mixed electronics parts for examples (LEDs, Resistors, Potentiometers, Buttons)
- Microsoft Windows 7 and above and Mac, or Linux OS
- Arduino IDE

Which Board Do I Use?

There are many revisions of the Arduino board. We will go over a couple of key contrasts between the boards.

The Basics: These boards are useful for beginning with and proper for basic programming.

Arduino Uno: It is the first board for the beginners with 14 digital input/outputs (6 of which can be PWM pins), six analogical inputs of info, and a 16MHz clock **.** It enables it to interface with an assortment of sensors and applications. The Uno works at 5V and can get control using the USB port, or it can take an information voltage of 6-20V (7-12 advised) using pin in. The 5V pin can supply 1A max for driving external sensors and outputs on the off chance that increasingly, at that point that is required you need an external power supply. Here attempt to draw frequently; at that point 1A, the board will get harm. This device has a measurement of 2.7" by 2.1".

Arduino Micro: The Arduino Micro is a small control board for when space is at best in class in your project. It contains 0.7" by 1.9". The Arduino Micro has a large number of specs particulars from the Uno. The serial communication with the PC complete in an unexpected way, see the Arduino Leonardo board for more features.

Arduino Pro/Pro Micro: It is a pro board, comes in various variants. The Pro is a full sized board, perfect with the multiple safeguards. The Pro Mini measures 0.7" by 1.3,", which makes it ideal for little tasks. Each board arrives in a 5V and 3.3V version. There are no headers appended to the board, which makes it somewhat harder to use for prototyping, yet gives it greater adaptability when you are joining the board into your project. The Arduino Pro has a considerable lot of indistinguishable particulars from the Uno, with a couple of differences. The 3.3V version keeps running on an 8 MHz clock; the 5V has a 16MHz clock. For power input, you need 3.3-12V for the 3.3V board and 5-12V for the 5V board.

Arduino Nano: The Arduino Nano is utilized for when you need the size of an Arduino Micro, yet the usefulness of an Uno. Fourteen digit i/o pins (6 can be PWM), "8" analog inputs, and a 16MHz clock. It keeps running on 5V, taking an information control voltage of 7-12V. It measures 0.73" x 1.7".

More developed Boards:

These Boards have further developed features for your next dimension projects.

Arduino Mega: The Arduino Mega can use for undertakings with vast amounts of data sources and outputs. Every one of the specs is equivalent to the Arduino Uno, yet it has 54 digital inputs/outputs (14 can be PWM), 16 analog inputs, and 4 UARTs for serial communication. It measures 2.1" by 4".

Arduino Leonardo: The Arduino Leonardo has the same specs from the Arduino Micro. The stark contrast between the Leonardo/Micro and the various boards is that there is no external chip for USB programming. Everything incorporates with one controller. It enables these boards to interface through a "Virtual COM port," and permits the Leonardo/Micro to convey as a keyboard/mouse to the PC. It additionally implies that not at all like different boards, there is no reset when the serial port opens. For debugging your projects, you have to do workarounds to see the Serial.prints() in your setup() schedule.

Arduino Due: The Arduino Due is the greatest and most exceedingly critical of the sheets. The most significant contrast between this board and the others is that it keeps running at 3.3V, not 5V. It indicates that you may require external circuitry to interface this with conventional 5V sensors. You can, in any case, input 7-12V, yet the input & output pins need 3.3V. The "Arduino Due" is the volume of a uber, 2.1" by 4". It has 54 digital inputs & outputs (12 can be PWM), 12 analog inputs sources, and 4 UARTs for serial communication. It keeps running on a 32-bit processor, running at 84MHz. It enables it to accomplish progressively necessary calculations (4 bytes at any given moment, rather than 2), more than multiple times quicker than the other Arduino loads up. It makes the applications keep running up to 10x faster. Due besides has a large number of interfering with pins, because of the second processor, enabling it to get significantly more criticism from its world around it.

Arduino Ethernet: The Arduino Ethernet board resembles an Arduino Uno with an Ethernet shield incorporated with it. There is additionally a small scale SD card reader built-in. It has a similar number of data

inputs/outputs as the Uno. However, four of them can use for communication with the Ethernet system, and four of them apply for the SD reader. It leaves six accessible digital input/outputs (4 can be PWM), and six analog inputs. By utilizing the ethernet port, you can associate this board to the web to open an entirely different world of possibilities for your plans.

Arduino Yun: The Arduino Yun is a remarkable board. It is a mixture of two CPUs. The first processor drives the Arduino programs, and the other processor runs a Linux appropriation that enables you to compose shell and python contents. Second, it has worked in WiFi and Ethernet abilities to make it simple to interface your tasks to the web. At long last, it has a worked in SD card per user to grow the new area about your programs. This board contains 20 input and output pins, in these seven(7) can be PWM and 12 analog inputs with a 16MHz clock. It quantifies 2.09" by 2.87".

Since we've gone over all the Arduino boards, it's an ideal opportunity to proceed onward to setting up the software!

Install the IDE after Downloading it

From the official Arduino website, you can download IDE. Since the Arduino utilizes a USB to serial converter (which enable it to interact with the host PC), the Arduino board is perfect with most PCs that have a USB port. Most certainly, you will require the IDE first. Fortunately, the Arduino designers have delivered various variants of the IDE for different working systems, including Windows, Mac, and Linux. In this instructional exercise, we will use Window 10, to guarantee that you download the right form of the IDE on the off chance that you don't have Windows 10.

Get the Arduino COM Port Number

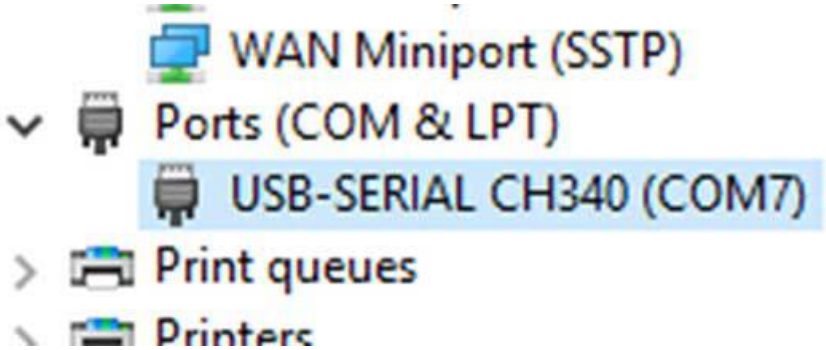
Next, you'll have to interface the Arduino Uno board to the PC. You can do it using a USB B association. Gratitude to the lovely universe of USB, we don't have to give the power to the Arduino, as the USB gives 5V up to 2A. At the point when the Arduino is attached, the operating system ought to perceive the board as a generic COM port (to clear the things, my Arduino Uno uses a CH340G, which is an RS-232 serial to USB

converter). When it's identified, we should discover what port number it has. The most straightforward approach to do It is to type device manager" into Windows Search and select Device Manager when it appears.

Note: ***Finding the device manager option in Windows 10. Right-Click*** → **My Pc → Properties → Device Manager**

In the Device Manager window, search for a device under "Ports (COM and LPT)," and chances are the Arduino will be the primary device on the menu. In my Device Manager, the Arduino appears as COM7 (CH340 is my device name).

COM7 (port 7), select appropriate yours.



Be cautioned; the Arduino won't generally recognize automatically. If your Arduino rejects it, at that point uninstall the driver, expel the Arduino, reinsert the Arduino, locate the unrecognized device, right click "Update driver," and after that click "Search automatically." It should fix 99% of your updating issues.

Occasionally, you need to update the drivers, in case, the Arduino not appropriately recognized.

USB-SERIAL CH340 (COM7)

Right-click → *update driver → Search automatically for updated driver software*

Windows sometimes can disturb you while using COM ports, as it can mysteriously change their numbers between connections. At some point, your Arduino might be on port 7 (as appeared), yet then on different days, Windows may move it to an alternate port number. So be

ready for this kind of stuff. As I get it, It happens when you interface other COM ports to your system (which I often do).

Along these lines, if you can't discover your Arduino on the port that you usually use, to check your port, go-to-your Device Manager, it is fundamental, update your driver

Image the IDE

Since we have discovered the COM port that the Arduino is on, it's an ideal opportunity to load the Arduino IDE and image it to utilize a similar device and port. Begin by stacking the IDE. When it's loaded, explore to Tools > Board > Arduino Uno. In case, if you are using an alternate board (i.e., not the Arduino Uno), you should choose the best possible board!

Order the IDE which board you are working here. The outlook of the software would be like It

Go to: Tools → Board Arduino/Genuine Uno → Arduino/Genuine Uno

Next, you should tell the IDE which COM port the Arduino is on. To do It, explore to Tools > Port > COM7. On the off chance that your Arduino is on an alternate port, select that port.

Go to: Tools → Port "COM7" → COM7

Loading a Basic Example

For the sake of simplicity, we will charge an example project that the Arduino IDE comes with this. Its example will make the onboard LED blink for a second continuously.

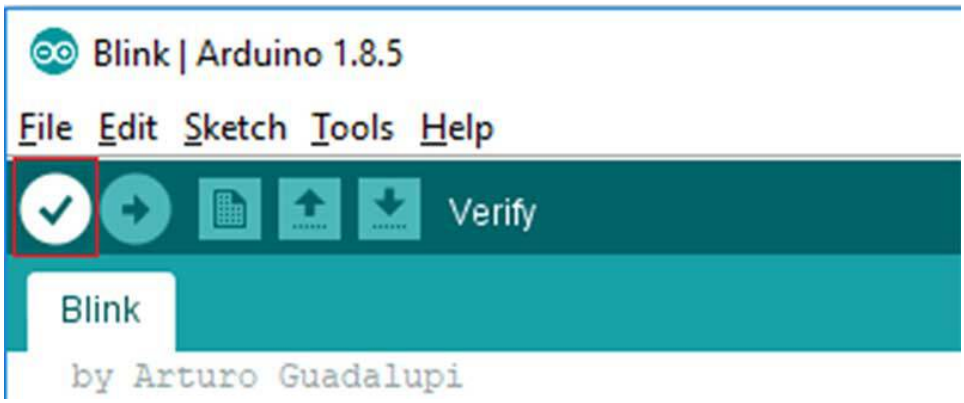
Load the blink example.

To load the example, *Go to File* → Examples → 01.Basics → Blink

With the model loaded, it's an excellent opportunity to check and upload the code. The verify stage checks the system for errors; at that point aggregates the prepared for transferring code to the Arduino. The upload stage takes the binary data, which made from the system and uploads it to the Arduino using the serial port.

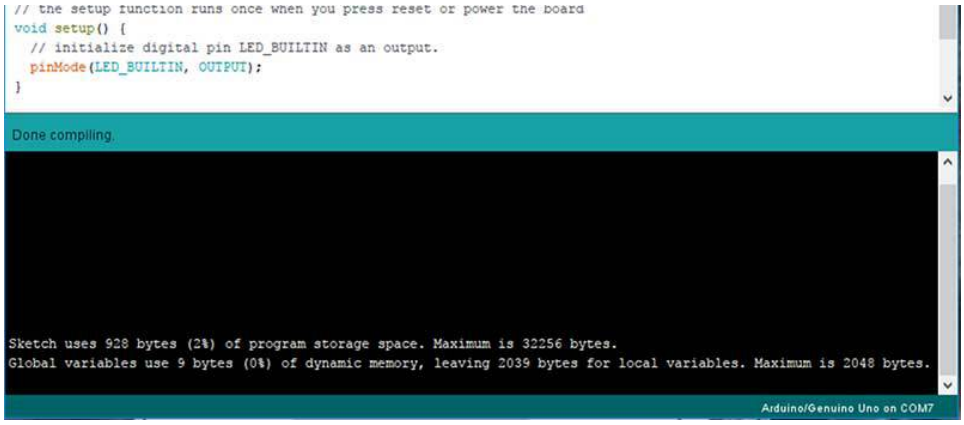
To confirm and organize the code, press the check mark button in the upper left window.

The "Verify" button will order the Arduino code.



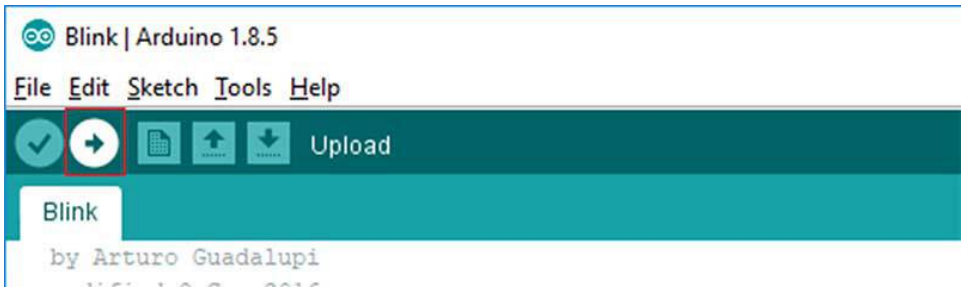
On the off chance that the compilation stage was active, you should see the accompanying message in the output window at the base of the IDE. You may likewise observe a similar note—only it's one that does not have words like "error" and "warning."

It is a progressive accumulation.



With the code incorporated, you should now transfer it the Arduino Uno. To do It, click the arrow by the check mark.

The "Upload" catch will program the Arduino with your code.

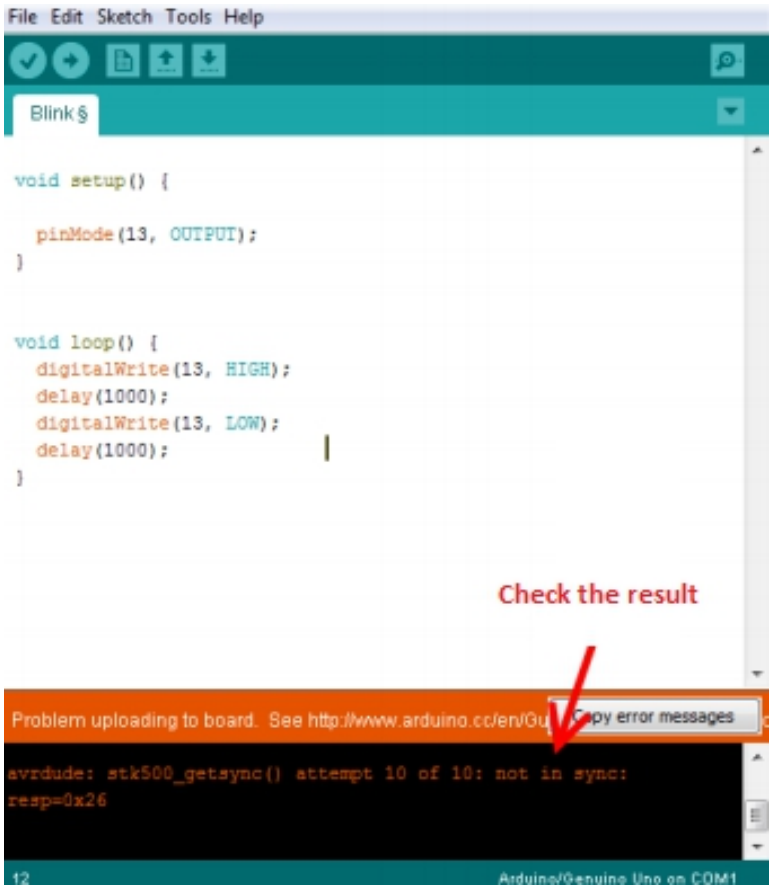

Blink | Arduino 1.8.5
File Edit Sketch Tools Help
Upload
Blink
by Arturo Guadalupi

Chapter 4

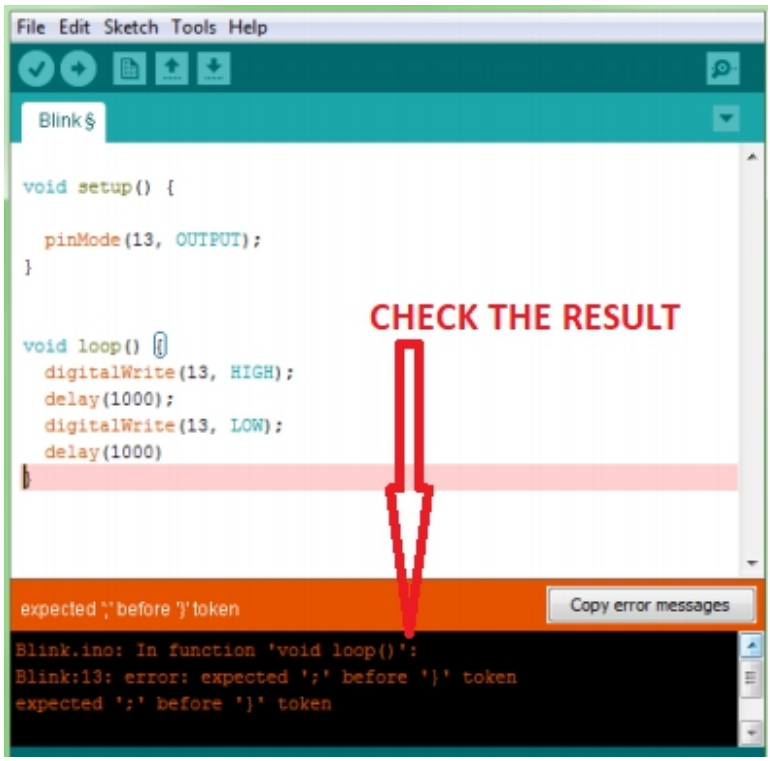
Coding for the Arduino

Above all else, a brief clarification for conceivable error reports that can show up while working with the Arduino software. The two most normal ones are:

1) The installation of the board is not correct or the wrong board selection. After uploading the sketch, there will show up an error report underneath the design. It would appear that the one in the image on the right. The note "not in sync" appears in the error report.


File Edit Sketch Tools Help
Blink §
void setup() {
pinMode(13, OUTPUT);
}
void loop() {
digitalWrite(13, HIGH);
delay(1000);
digitalWrite(13, LOW);
delay(1000);
}
Check the result
Problem uploading to board. See http://www.arduino.cc/en/Gu
Copy error messages
avrdude: stk500_getsync() attempt 10 of 10: not in sync:
resp=0x26
12
Arduino/Genuino Uno on COM1

2) There is a slip-up in the sketch. For instance, a word spoke is incorrect, or a bracket is missing. In the precedent on the left, the last semicolon in the plan is dropping. In this Case, the error report frequently begins with "excepted..". It implies the program is as yet expecting something missing.



The fundamental structure of a sketch:

A sketch can divide into three sections.

1. Name variable

In the first segment, elements of the program name (This will be clarified in the program SOON). This part isn't completely vital.

2. Setup (entirely vital for the program)

The structure will perform just once. Here you are telling the application for instance what Pin (slot for cables) ought to be input and what ought to be output on the sheets.

Defined as Output: The board should put out a voltage. For instance: With this switch, a LED is intended to illuminate.

Defined as an Input : The board should peruse out a voltage. For instance, A switch will incite. The board perceived this since it gets a voltage on the Input pin.

3. Loop (essential for the program)

The board will continuously rehash this loop part. It absorbs the sketch from start to finish and begins again from the beginning and so on.

Coding a Voltmeter

LEDs have a voltage drop of around 2 volts over their central intersection, and the voltage can differ contingent upon the size, shade of the LED, and forward current. We will utilize a single contribution to the Arduino and measure the precise estimation of the voltage drop. Since the microcontroller works like a computer, the simple data must be changed over into the parallel framework to handle. By and large, when we go from simple to advanced, we utilize a device called a simple to-computerized converter (ADC). Going the other way, originating from the computerized domain to the real world, we employ an advanced to-simple converter (DAC). For us to gauge a simple voltage, rather than a rational level, the Arduino ADC will change over the vitality to a binary number somewhere in the range of 0 and 1,023.

The number 1,023 is the maximum voltage that the processor can handle, which equates to 5 volts with the UNO. If you consistently measure a voltage lower than 5 volts, use the analog reference pin AREF to increase the ADC accuracy.

A reference voltage can quickly develop across a resistor voltage divider. Our project code, shown in Listing 8-1, uses the entire range and reads the voltage once each second, and then displays the ADC value between 0 and 1,023, equating to the actual voltage value of between 0 and 5 volts. For finding the actual voltage from the ADC, in our case we divide five by the 1,024 steps for the conversion factor (5/1,024 = 0.004883), then the real energy is found by multiplying the ADC step number reading, by the conversion factor, which is the height of each step. Because we want a precise voltage value, rather than using the integer data type, we declare variables as a float for floating point decimals. Voltage is also described as

a difference of potential and measured across a component. Because our Arduino voltmeter is referenced to ground potential, though, we can only directly check across the bottom element in a circuit, so we will build the LED circuit shown in Image 8-1 and connect our analog input pin between the LED and the resistor. We happened to pick badge

A0 as our input pin, but the UNO has six analog pins and can use any of them. They can also be used for digital I/O when needed.

Listing 8-1. **Coding a Voltmeter**

```
const int voltageIn = A0; //Makes a volt meter up to 5 volts
int voltLevel;

   float actualVoltage;
void setup (){
Serial.begin(9600);
pinMode (voltageIn, INPUT);
}
void loop(){
voltLevel = analogRead(voltageIn);
actualVoltage = (voltLevel * .004883); //volt steps are .004883
Serial.println();
Serial.print("The level is ");
Serial.println (voltLevel);
Serial.print("The actual voltage is ");
Serial.println (actualVoltage);
Serial.println();
Serial.println();
delay(1000);
}
```

After running the code for the voltmeter, the ADC number representing 2 volts is 410 with both names being displayed on the serial monitor if you measured a typical red LED; however, the values could vary due to the type of LED and component tolerances. Using Kirchhoff's voltage law, we can assume that because 5 volts is the total voltage, and we were reading 2 volts across the LED, there must be 3 volts across the current- limiting resistor. We can also measure the voltage across the resistor, but because the Arduino is taking a reading with the ground as the reference point, we must interchange the LED and the resistor in the circuit. After swapping the two components around and running the program, you should observe approximately 3 volts across the resistor. As one last measurement, we could reverse the LED and measure zero volts across the resistor. Because the LED and resistor are in a series circuit and the current loop broke, no current will flow through the resistor to produce a voltage drop across

it (remember Ohm's Law). The current is near zero because the LED is

A polarized component and the current can only flow in one direction through a diode. There is a negligible amount of reverse current, but if

the input voltage were to increase drastically, a breakdown of the LED junction could occur and cause a short circuit. There is no voltage across a short circuit and a maximum voltage across an open circuit. We can simulate

an open LED by orienting the two components into their initial positions, as shown in Image 8-1 . Then with the resistor on top and the LED on the bottom, reverse the polarity of the LED by flipping around so that the negative side is facing toward the positive voltage source. The LED should be off, and the voltmeter will read 5 volts across the open circuit. It illustrates two valuable indications for troubleshooting circuits: Shorts have zero volts across them, and there will be the maximum voltage across an open circuit. With the LED reverse biased, it is essentially a free point in the course.

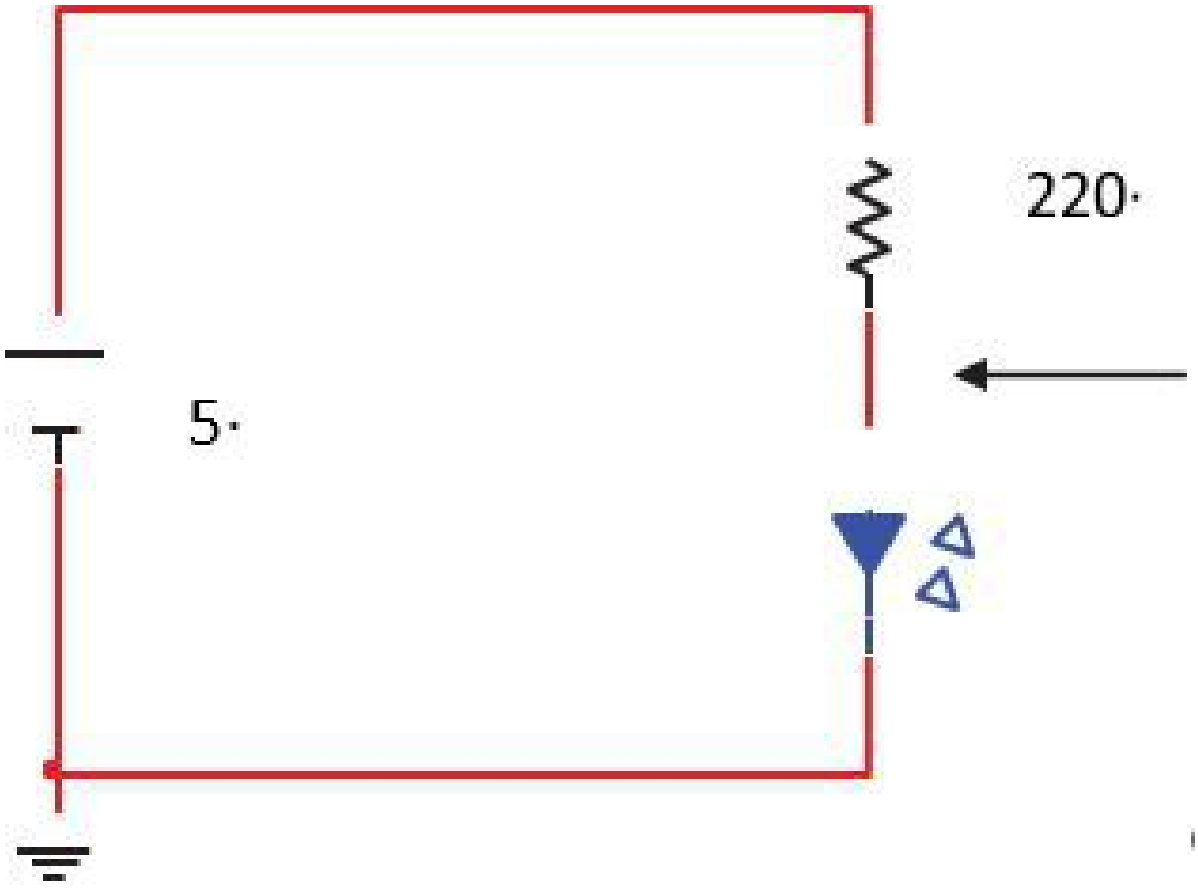


Image 8-1. An LED circuit

Dimming an LED with Pulse Width Modulation

In the last project, we used an analog voltage as an input to the microcontroller. It would be useful in a variety of applications where analog sensors are used to either produce varying energy or resistance as conditions change. In this project, we will create a pseudo-analog output coming from the microcontroller. Although a digital device can only approximately generate a real analog production, because there are voltage level steps involved in the DAC process just as we saw with the ADC, the steps can make small to approach a real analog signal output. There are two ways to produce an analog output: One way is to build a varying voltage with the DAC, and the other is by varying the pulse widths of square waves. In a few of our earlier projects, we rapidly flashed an LED and noted that its brightness appeared steady but dim. This pseudo analog technique is called pulse width modulation (PWM). In digital electronics, there are two logic levels. For the Arduino UNO, a low level is zero, and a high is 5 volts. A high pulse is shown between the two arrows in Image 8-2, with a weak pulse shown immediately to the right. Both the high vibration and low pulsation make up one cycle. In Image 8-2, we Chapter 8 Electronic Projects 161 would say that the signal is a square wave because the pulse widths are equal. The time one is the same as the time off. It has a 50% duty cycle. When the times of the periods are short, and the flashing is fast, our eyes do not discern the flashes, but instead, we will notice that the overall brightness decreases. The characteristic of our vision called persistence allows us to watch television and movies and not see any flickering between frames. If the high pulse time shown in Image 8-2 were to lessen, the duty cycle would go down, and the LED would appear dimmer because the entire cycle time would remain the same, but the LED would be on for a smaller fraction of the total. It explains how PWM works with vision, but PWM can also use as a way to modulate communication signals and to control motor speed.

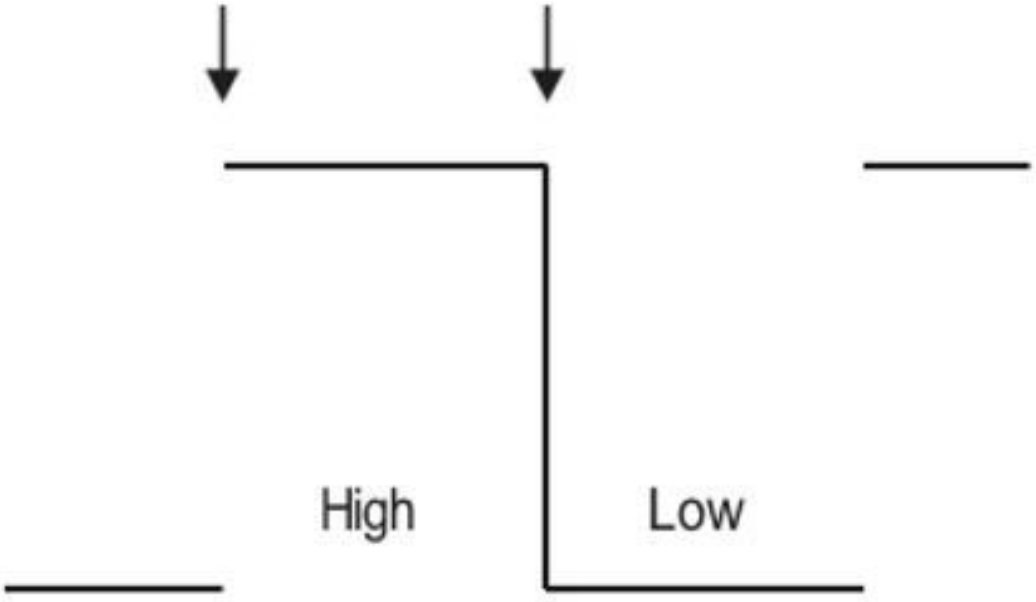


Image 8-2. *Pulse width modulation*

The Arduino has several pins for PWM; on the UNO they are 3, 5, 6, 9, 10, and 11. This project uses PWM to control the brightness of an LED. Start by connecting a 220 Ohm resistor and an LED between pin 9 and ground. After loading the code in Listing 8-2, we can control the LED intensity by momentarily grounding any one of three pins. We use pin 7 for 100% duty cycle (full brightness), pin six drops to medium light, and pin five drops the LED to low light. The command analogWrite identifies the output, and the number that follows is the PWM duty cycle broken up into 256 steps, 0 to 255, with 255 being the highest duty cycle producing full output. We picked the average value to be 50% duty cycle, and the low to be about 25% duty cycle.

Listing 8-2 . Dimming an LED with PWM

```
const int LED = 9;   // A three intensity LED program
const int highPin = 7;
const int medPin = 6;
 const int lowPin = 5;
boolean high;
boolean med;
 boolean low;
void setup(){
pinMode (LED, OUTPUT);
pinMode (highPin, INPUT_PULLUP);
pinMode (medPin, INPUT_PULLUP);
pinMode (lowPin, INPUT_PULLUP);
}
void loop(){
high = digitalRead (highPin);
med = digitalRead (medPin);
low = digitalRead (lowPin);
if (high == LOW){
analogWrite(LED, 255);
 }
if (med == LOW){
 analogWrite(LED, 125);
 }
 if (low == LOW){
 analogWrite(LED, 60);
 }
 delay(100);
```

Controlling an LED Using a Light Sensor

In the schematic in Image 8-3 , we use a photoresistor to vary the conduction of an NPN transistor circuit used to illuminate an LED.

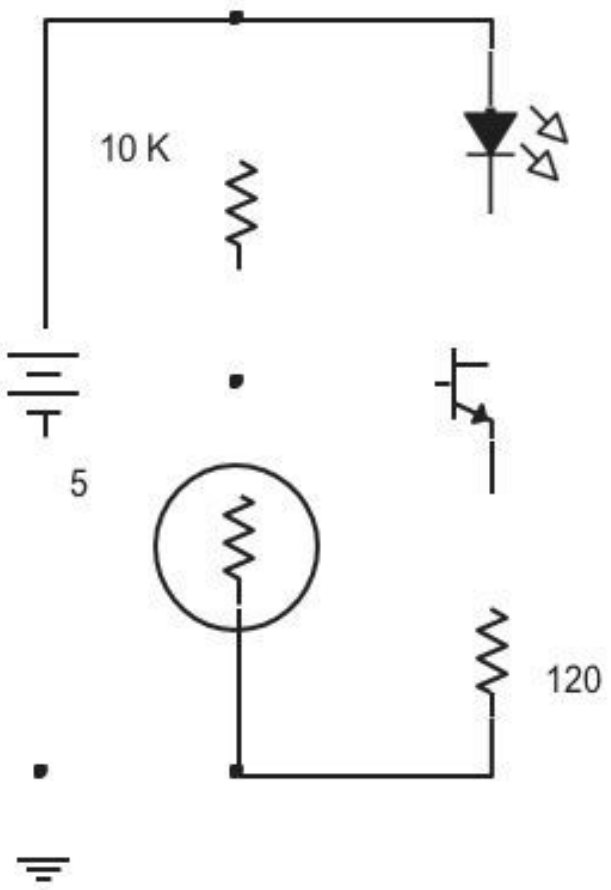


Image 8-3. Transistor circuit used to illuminate an LED

The photoresistor, as shown in the circle, exhibits decreasing resistance as light increases. There is a voltage at the point between the two resistors that also connects to the control element of the NPN transistor, called the base. If there is decreasing resistance across the

photoresistor caused by increasing light, the corresponding decreasing voltage also appears on the transistor base. If the base voltage drops, it produces a less base current through the transistor, which causes the LED to have less current through the vertical section of the transistor's emitter and collector sections, which therefore causes the LED to go dimmer, or

off (i.e., more ambient light, less LED light). Conversely, if there is less light on the photoresistor, the resistance goes higher, the voltage goes

more elevated, and the current through the LED goes higher, causing it to get brighter (i.e.,

Less ambient light, more LED light). The objective is that in bright ambient light, the LED is off, and in low light, the LED is on. If the light changes are somewhat gradual on the photoresistor, the electronic circuit will produce a slightly progressive change in LED intensity. If parts are available, build, and test the course.

We will use the Arduino to switch an LED on and off. It exercise (Listing 8-3) could easily be adapted to control outdoor lighting, security systems, and other devices for which operation is dependent on differentiating day from night. To add the microcontroller, connect a

wire from the intersecting point between the two resistors and transistor base to the Arduino analog input pin A0. We are using an analog input because the voltage developed across the photo resistor will vary in an analog manner related to the amount of light intensity. (You might also wish to modify later the program to output a PWM signal to change the LED brightness, but our plan is only interested in sensing between light and dark, and correspondingly switching an LED off or on. Please modify the code to achieve a different switching response, and you might need to do It, as ambient light conditions will vary.) The analog voltage developed across the photo resistor is represented by several from 0 to 1,023. We could scale our input for more accuracy by using an external reference connected to the AREF pin on the Arduino and adjust the code accordingly. In our example, we use the ADC number of 1,024 to represent the full 5 volts. (Again, keep in mind that you might need to make adjustments to the analog read names we call lowLight in Listing 8-3 , depending on the amount of light intensity you are working in.)

Listing 8-3. Controlling an LED with a Light Sensor

```
const int LED = 13;
    int lowLight;

    boolean on;
    void setup(){//analog pins are inputs by default
      pinMode (LED, OUTPUT);
      Serial.begin(9600);
      Serial.println("the serial monitor is displaying the light value");
    }

void loop (){
      lowLight = analogRead (0); //reads voltage at 1024 steps
      if (lowLight > 140) { //ambient light low, turn on LED
        digitalWrite (LED, HIGH);

        on = true;
      if ((lowLight <= 140) && (on == false)){
        digitalWrite(LED, LOW); // ambient light high, turn off LED
      }
      if ((lowLight <= 120) && (on == true)){

      //hysteresis

        digitalWrite(LED, LOW);
      }

      Serial.println (lowLight);
      delay (500);
```

In the bottom section of code where we commented about hysteresis, without adding that section, the controller could have possibly flickered the LED when the ambient light was very near the switching threshold.

Hysteresis is used to lock in a function until there is a significant change in the input. It is used in thermostats so that the heating or cooling periods are distinct, so that temperature control units do not repeatedly cycle on and off near the set point.

Analog sensors tend to be a little finicky, so the code that we presented might need to be adapted to your specific lighting conditions and breadboard circuit build. That is why we added the serial monitor function into the code. After uploading the system, you can open the serial monitor and check the analog read as you expose the photoresistor to light and dark conditions, and then adjust your numbers for the proper switching function. Once the Arduino code is working correctly, you will notice that the transistor circuit varies the intensity of its LED in an analog manner, whereas the controller abruptly switches the onboard LED near pin 13 on and off.

Analog circuits can also abruptly switch logic levels. It can complete in several ways, with one solution being the use of a hybrid device called a *comparator,* which is an open-ended operational amplifier

(op-amp), which can image as a high-gain voltage amplifier. With small changes near the switching point, it can jump from rail to rail, between low and high levels. Comparators are available as ICs. On the other hand, microcontrollers can simulate analog outputs, as with the PWM project that we looked at in the last section. With the inclusion of a DAC, they can mimic analog circuit output, as in the case of a CD or MP3 player. There are gray areas between analog and digital technology.

Coding a Frequency Counter

The schematic showed in Image 8-4 to demonstrate how a process could be implemented either by using discrete physical components or by creating a program to have a microprocessor perform the procedure. The process that we are examining is a free-running astable multivibrator. Due to the resistive and capacitance components connected to the NE555 (LM555) timer, a square wave output observes on IC output pin 3, which flashes an LED once a second. (The production is approximately one half-second off and one half-second on.) Timing circuits like It are useful in providing clock pulses for timing purposes; however, the accuracy of a 555 timer circuit is not very good due to the wide tolerances in resistive and capacitance components. Standard resistors have a plus or minus tolerance of 5%, and capacitors have an even Wider understanding. If accuracy were an issue, a hardware solution could be a crystal controlled oscillator, and such ICs are readily available.

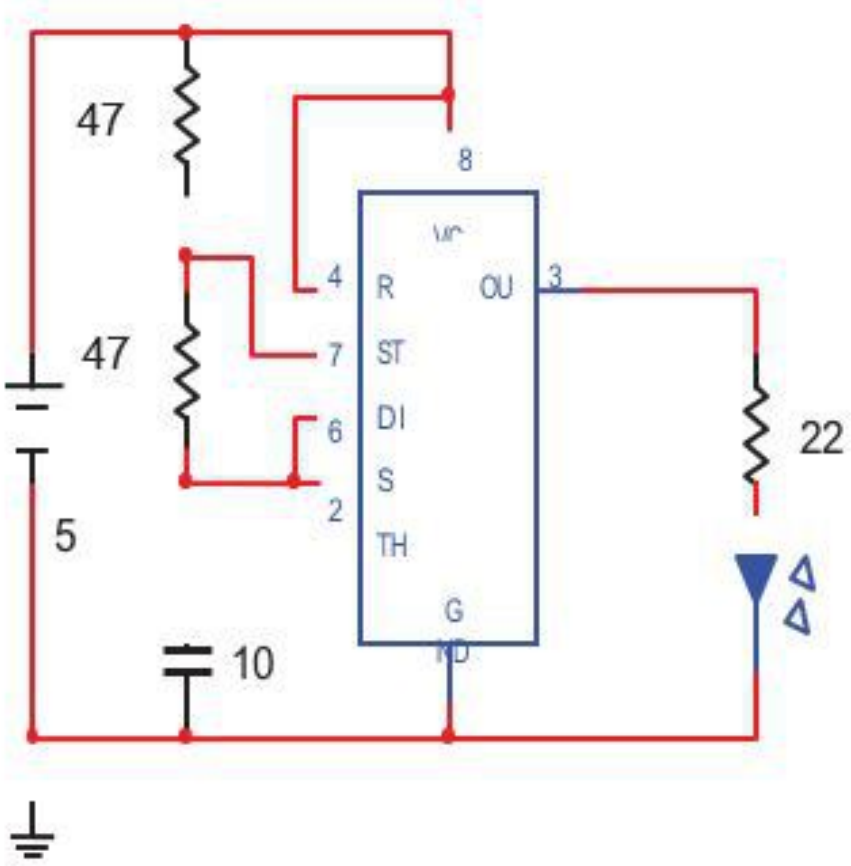


Image 8-4. A timing circuit

Our next project is to use the 555 timer IC as a square wave frequency source and create a program to read its frequency and generate an output on the Arduino, indicating that the rate is within tolerance. If the electronic components are not available; you can use a function generator or a second Arduino to act as the frequency generating device, as described in the next section.

To power the device, we are connecting the VCC power line of the 555 to the 5-volt header pin on the Arduino to use USB power and the ground line to one of the Arduino ground pins.

Use pin three(3) of 555 should be connected to pin three on the Arduino; the LED circuit can remain connected on the breadboard. We use pin three on the Arduino just because of the excellent number match, but any digital I/O pin will work just as well (see Image 8-5).

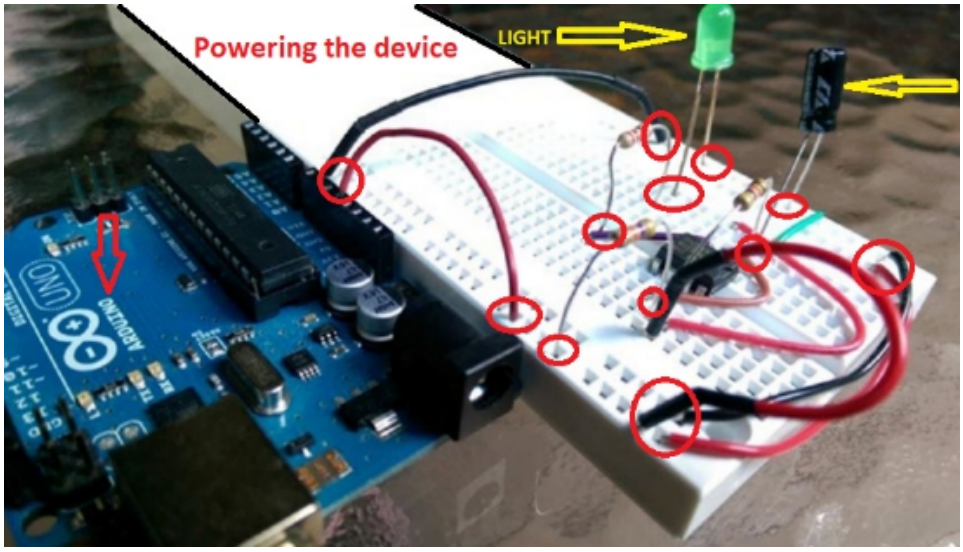


Image 8-5. Powering the device

The breadboard has a gap in the middle, separating two similar sides. The holes on each side running parallel with the short edge of the breadboard are all connected. Usually, there are five holes in a row on each side, and they are related, but there is not a connection to the holes on the other side of the gap or any other breaches. Running perpendicular along the long side of the breadboard are two parallel lines of trenches. Each parallel line running along the length of the board is connected, but they are not connected anywhere else. These two sets of parallel lines are mainly for use as power buses and can each be jumped together to the collections of lines on the other side of the board. The IC is placed in any convenient location straddling the middle gap. It can check in Figure 8-5 that the breadboard is getting 5-volt power and ground from the Arduino, which has a connection to a computer via the USB connector. As mentioned, also attach a wire between pin 3 of the IC on the breadboard and pin three on the Arduino. If built, as shown in Figure 8-4, the 555 will generate one pulse per second, which equates to a frequency of 1 Hz. The code in Listing 8-4 will read the square wave pulses from the Arduino and display the number 1 on the serial monitor, showing that it is counting a 1 Hz signal.

Listing 8-4. A Frequency Counter

```
//freq counter reads sine, square, and triangle waves
unsigned long currentMillis;     // works from 1 Hz to 20 Khz
unsigned long lastMillis;        //samples once a second
unsigned long duration;
int pulseHigh;
int pulseLow;
long halfCycles;
long cycles;
int in = LOW;
void setup(){
  pinMode(3, INPUT);
  Serial.begin(9600);
  Serial.println("The frequency is displayed in Hertz");
}

void loop(){
  lastMillis = millis();
  do{
    in = digitalRead(3);
    currentMillis = millis();
    duration = currentMillis - lastMillis;

    if (in == HIGH){
      pulseHigh = 1;
    }
    if (in == LOW){
      pulseLow = 1;
    }
    if (pulseHigh ==1 && pulseLow == 1){
      halfCycles = halfCycles + 1;
      pulseHigh = 0;
      pulseLow = 0;
   }
  } while(duration < 1000); //looks for cycles per second

  cycles = halfCycles / 2; //divide by two, there are two half cycles
                           //in a cycle
  Serial.println(cycles);
  cycles = 0;
  halfCycles = 0;
  delay (1000);
}
```

The do-while loop in the code is using the millions timer to count an elapsed time of 1 second, as the frequency can define as cycles per second (Hz).

The pulse count is triggered as the waveform goes through one alternation between positive and negative value, as shown in Figure 8-6 , and then again for the low pulse between the negative going transition and the next positive going transition. Because It trigger point accounts for one half of the waveform and occurs twice for each cycle, the number is divided by two to report the actual frequency in cycles per second (Hz). The code will work for square waves, sine waves, and triangle waves. If a function generator is available, it would be an exciting project to try all three waveforms at different frequencies up to 20 kHz. Care must be taken, however, so that the waveforms are in the range between 0 and 5 volts

DC. Use of a TTL output is convenient.

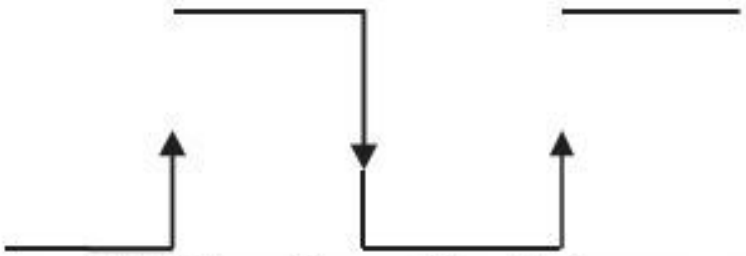


Image 8-6. The alternation between positive and negative values

We now change our 555 circuit's frequency by removing and replacing the ten µF capacitor with a 0.1 µF capacitor to increase the rate from about 1 Hz to 100 Hz. The 0.1 µF cap is nonpolarized, meaning that there is no polarity consideration for positive and ground. Small capacitors under one µF are generally nonpolarized. Because of their small physical size, a code is sometimes used to identify the value of small capacitors.

The first number represents the first digit, followed by the second digit, with the third number representing the number of zeros, and the total value as picoFarad in engineering notation. Our 0.1 µF cap will have the

code 104, as 1 and a 0 followed by four more 0s means 100,000 pF, and that is equal to 0.1 µF. (If you have a question about It, be sure to review the engineering notation information earlier in the text.)

The adaptation to the previous program will display the exact frequency of the 555 on the serial monitor as before; however, the code is now slightly enhanced to additionally flash the onboard LED connected to pin 13 if the frequency is within plus or minus 10 Hz of

100 Hz. Lighting the LED could also be used in a real-world application of checking for a good signal condition. Additionally, the code, or the circuit, could be modified to flash a lamp or sound a buzzer if the signal goes

out of tolerance. The changes to the system to now respond to the correct frequency are shown as highlighted in Listing 8-5.

Listing 8-5. Responding to the Correct Frequency

```
unsigned long duration;
  unsigned long currentMillis;
  unsigned long lastMillis;
  int pulseHigh;
  int pulseLow;
  long halfCycles;
  long cycles;
  int in = LOW;
  const int LED = 13;

  void setup(){
    pinMode(3, INPUT);
    Serial.begin(9600);

    pinMode(LED, OUTPUT);
  }
 void loop(){
      lastMillis = millis();

do{
        in = digitalRead(3);
        currentMillis = millis();
        duration = currentMillis - lastMillis;


if (in == HIGH){
          pulseHigh = 1;
        }
        if (in == LOW){

  pulseLow = 1;
        }
if (pulseHigh ==1 && pulseLow == 1){
          halfCycles = halfCycles + 1;
          pulseHigh = 0;
          pulseLow = 0;
        }
   while(duration < 1000);    //looks for cycles per second
          |

   cycles = halfCycles / 2;
    Serial.println(cycles);
  if ((cycles > 90) && (cycles < 110)){
  digitalWrite (LED, HIGH);
  }
   cycles = 0;
   halfCycles = 0;
   delay (1000);
 digitalWrite (LED, LOW);
 }
```

Pulse Generation

The Arduino can produce a frequency output, as was done previously with the 555 timer IC. The advantage of using code to control the frequency is that no IC, resistors, or capacitors need to use. Our “code” also allows for real-time control of the rate via text input to the serial monitor.

The frequency output is also displayed on the serial monitor and sent to a digital output pin to control an LED and an average speaker, or possibly a piezo speaker. A piezo speaker uses the piezoelectric effect to generate sound. The piezo element can create with a crystalline material that makes a sound as a varying voltage can place across the component. It does It by distorting (bending) as the voltage varies. The piezoelectric effect has reciprocity, as do most devices that act as a *transducer* . A transducer is a gadget that changes over one type of vitality to another. In our case, the element will convert electrical power to mechanical energy that will produce sound.

As a side note, we all know the law of conservation of mass and energy, which states that matter and energy cannot be created or destroyed, but can change from one form to another. Once we use electrical power to bend a crystal and produce sound or move a speaker diaphragm, then, what kind of mass-energy does the process ultimately create? (We will save that thought for a chapter review question.)

As mentioned, the piezoelectric transducer has reciprocity, which means that it bends when a voltage will apply across its element, and conversely it also can produce electricity when a mechanical force is used to turn it. It can use as either an input device (sensor) or an output device (actuator), as in our application. One advantage of using a piezo speaker is that they are inexpensive and lighter in weight than a regular speaker. Piezoelectric elements also have a capacitive effect and

capacitive devices pass higher frequencies better, so the higher rates will generally have better sound fidelity than will low frequencies. Typical human hearing has a range of from 20 Hz to 20,000 Hz (20 kHz), although as one age the frequency response to high rates is lessened (some people say that It accounts for long marriages). The lowest frequency we can produce with our code is 35 Hz, due to the

tone command limitation. On the high side, we can provide rates well above the range of human hearing. Because we are using the unsigned int type for the tone command (Listing 8-6), we can go as high as 65,535 Hz. The piezo speaker should give the most favorable results between 500 and 5,000 Hz. If you do not have a piezo speaker, you can connect a small regular 8 Ohm speaker to the Arduino. As shown in Figure 8-7 , the current can control by adding the series resistor as shown, or damage to the Arduino could result. We are also using the onboard LED near pin 13, so you can see the results even without a sound-producing device. Even at the lowest Arduino tone of 35 Hz, the individual flashes are not distinguishable, but the LED flashing results in reduced brightness. You will also notice that when an “x” is input in the serial monitor, the LED entirely extinguishes because the pulse generation output stops.

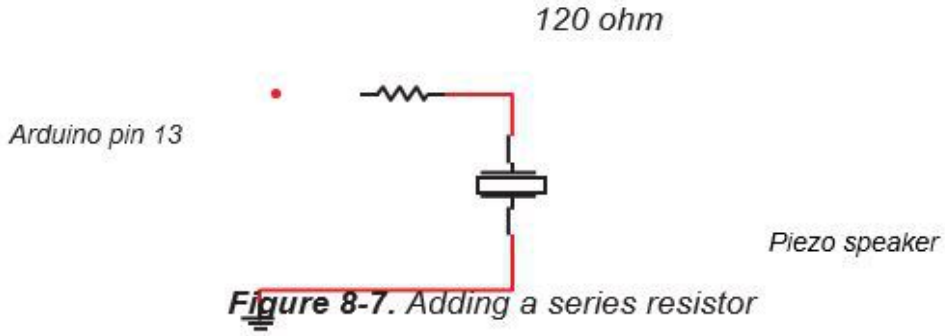


Figure 8-7. *Adding a series resistor*

Listing 8-6. Entering Frequency for Pulse Generation

```
unsigned int inputNumbers;
//code to enter frequency for pulse generation
unsigned int frequency;
String numbers = "";          // empty string for input numbers

const int LED = 13;
  Serial.begin(9600);
void setup() {
  pinMode (LED, OUTPUT);
  Serial.println("Enter a number from 500 hz up to 5000 hz");
  Serial.println();
  Serial.println("(You must put hz after the number, and then
press enter or click send.)");
  Serial.println();
}
void loop() {
    inputNumbers = Serial.read();
    if (isDigit(inputNumbers)) {
      numbers += (char)inputNumbers;

// now looks for the upper or lowercase ASCII code for h:
      //puts the numbers in sequence as string, converts to numbers
          Serial.print(numbers.toInt());
          Serial.println(" hz");
          Serial.println("To enter if incorrect, to end enter x");
          Serial.println();
          frequency = (numbers.toInt());

          tone (LED, frequency);
          //clears variables
 inputNumbers = 0;
          numbers = "";
    if ((inputNumbers == 120) ||  (inputNumbers == 88)){
          // upper or lower case x, shuts off tone function
          inputNumbers = 0; //clears variables
          numbers = "";
 Serial.println("idle, enter frequency for output:");
          Serial.println();
          noTone (LED);
     }
          Serial.print(numbers.toInt());
          Serial.println(" hz");
 if ((inputNumbers == 104) || (inputNumbers == 72)) {
   Serial.print("You entered a pulse frequency of: ");
          Serial.println("To enter if incorrect, to end enter x");
          Serial.println();
          frequency = (numbers.toInt());

          tone (LED, frequency);
          //clears variables
 inputNumbers = 0;
          numbers = "";
    if ((inputNumbers == 120) ||  (inputNumbers == 88)){
          // upper or lower case x, shuts off tone function
          inputNumbers = 0; //clears variables
          numbers = "";
 Serial.println("idle, enter frequency for output:");
          Serial.println();
          noTone (LED);}
```

We look for numbers from the serial monitor input and ignore other characters while determining the frequency that will help us with the tone command. The names are put together as string type data and then converted to an integer when the user inputs either letter h or H for Hertz. When the user inputs x or X, the output stops. It project outputs a series of pulses at a given frequency. Square waves have tremendous distortion and sound terrible as audio signals, but the vibrations we are producing in our project could have many different applications beyond that of just producing sound.

Chapter 5

Turn your Arduino into a machine

Arduino has been a debacle ever since it struck the market. Arduino has been a rewarding choice for students designing their first activities. This expansion in intrigue is making many people tinker with technology themselves. We arranged a list of Arduino extends in which you can turn your Arduino into various types of machines, that would be composed to structure. Arduino lying around or are wanting to get one, these should prove to be useful.

You only need an Arduino Board and other hardware

Arduino, into Machines (Projects)

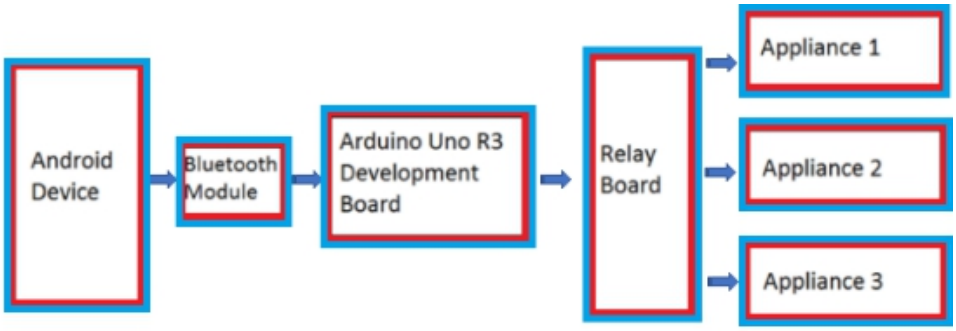
Home Automation Using an Android Device

This project can base on Interfacing an android application to Arduino Uno board using Bluetooth. The outcome is a home mechanization system with negligible electronic parts without convoluted welding and adaptable and bright design.

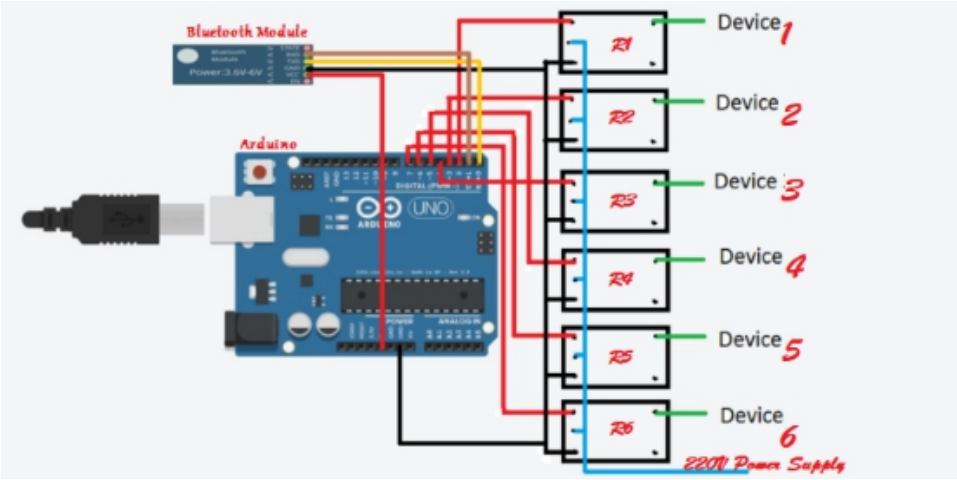
Parts List

- Arduino Uno R3 Development Board (or a believed working clone would work fine)
- Bluetooth Module (HC-05)
- Android Device (v4.0 or above)
- Relays (R1 – R6)
- Connectors
- For the Arduino, A USB

Square Drawing

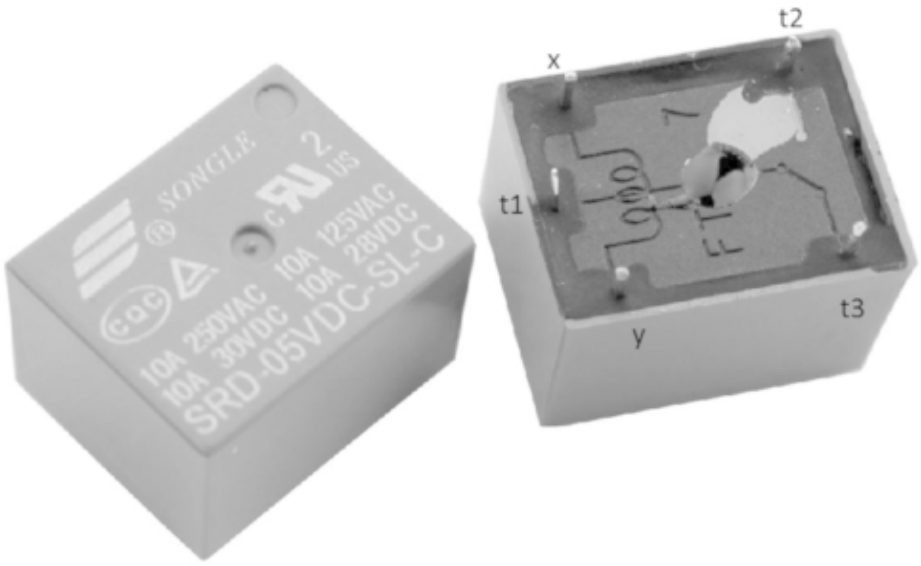


Blueprint Drawing



See, above image that can put upside down for straightforward reference to the relay pin connections presented in the following image.

Relay pin connections



Connections

- Interface pins 2-7 of Arduino to Relays R1-R6 at pin "x" of each relay separately.
- Interconnect all the "y" pins of each transfer and associate one of them to the GND pin of Arduino.
- Interface VCC of Bluetooth module to the 5v power pin of Arduino and in like manner Bluetooth GND pin to Module GND pin of Arduino.
- Interconnect all the "t1" pins of each transfer and interface one of them to 220V contribution of essential power supply.
- Interface any one terminal of every one of the devices to be controlled to pin "t2" of each of the transfers R1-R6 separately.
- Associate the other residual terminals of the considerable number of devices to be controlled to GND of the principal power supply.
- Now associate Tx of Bluetooth module to Rx of Arduino and Rx of Bluetooth module to Tx of Arduino.

Arduino Based Digital Clock with Alarm

It will be a digital clock with alarm, based on Arduino to indicate consistent utilizing an RTC IC DS1307 which tackles I2C protocol.

The continuous clock infers it seeks after even power failure. Right when power will reconnect, it demonstrates the consistent paying little respect to the time and duration; it was an off state. In this task, we have utilized a 16x2 LCD  module to show the time in the group. An Alarm selection can also include, and we can set up the alarm time. At the point when the alert time it saved in internal EEPROM of Arduino, it remains spared even after reset or power failure. There will be continuing standard checks in our PCs, houses, workplaces, and electronics device for keeping them stimulated with real time.

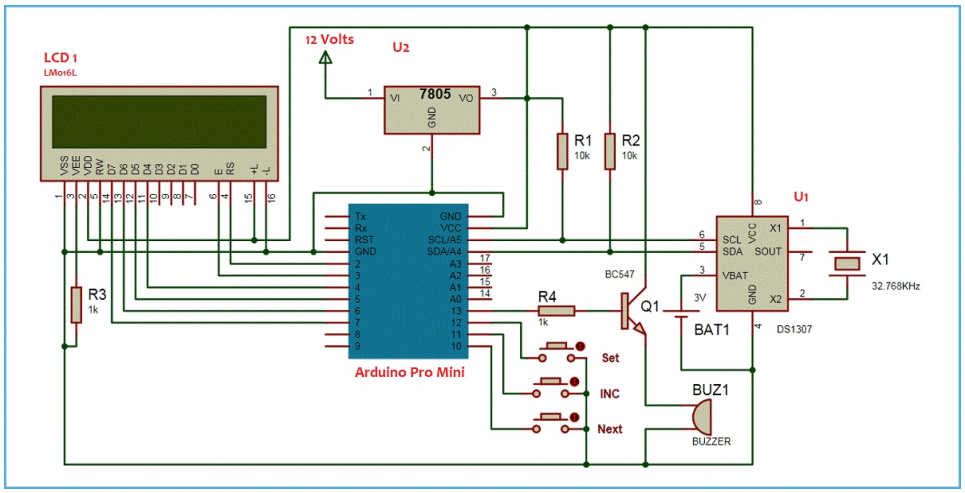
The I2C protocol is a technique to connect at least two devices utilizing two wires to a single framework. Thus this protocol is additionally called as two wire protocol. It can be used to communicate 127 methods to a single device. Most of the I2C tools run on100Khz frequency.

Actions for data writing master to slave

1) It sends start condition to slave.
2) It sends the slave address to the slave.
3) It “sends” the “compose” bit (0) to solve
4) It received ACK bit from the slave
5) It sends words “address” to solve
6) It received ACK bit from the slave
7) It sends information to the slave.
8) It received ACK bit from the slave.
9) And last, it sends stop condition to slave.

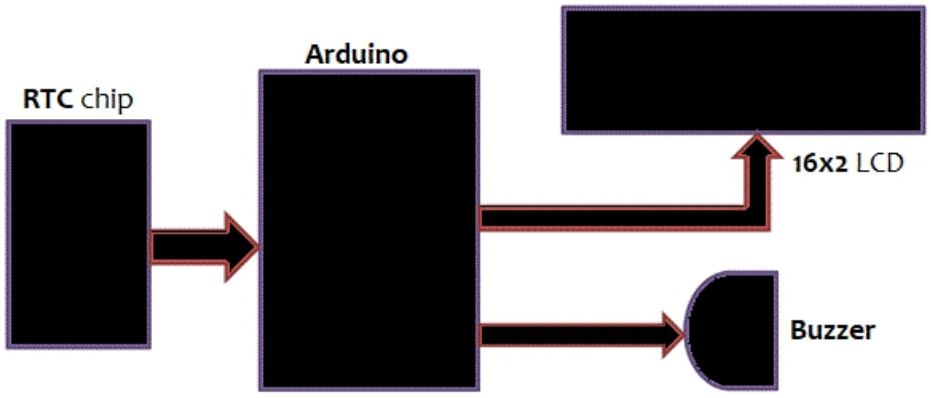
Start for data reading from slave to master

1) It sends START condition to slave.
2) It sends the slave address to the slave.
3) It sends read bit (1) to slave.
4) It received ACK bit from the slave
5) It gathered information from the slave
6) It received ACK bit from the slave.
7) It sends STOP condition to slave.



In this Arduino based digital clock circuit, we have used three significant components, which are IC DS1307, Arduino Pro Mini Board, and 16x2 LCD module.

Here Arduino is used for the understanding time from ds1307 and show it on 16x2 LCD. DS1307 sends time/date using two lines to Arduino. A ringer can also apply for an alert sign, which signals when the caution activates. A block diagram is here to comprehend the working of this Real Time Clock.



As should be evident in the circuit diagram, DS1307 chip pin SDA and SCL are connect with Arduino pins SDA and SCL with pull up resistor that holds default value HIGH at data and clock lines. A 32.768KHz crystal oscillator attaches with DS1307chip for producing definite 1-second delay, and connect an additional a 3-volt battery with pin 3rd(BAT) of DS1307 which keeps time running power failure.16x2 LCD is associated with Arduino in 4-bit mode. Control the pin RS, RW, and En

can directly connect with Arduino pin 2, GND and 3. What's more, information pin D0-D7 is associated with 4, 5, 6, 7 of Arduino. A ringer is associated with Arduino PIN 13 through an NPN BC547 transistor having a 1 k resistor at its base.

Three buttons in particular set, INC and Next, are utilized for setting the caution to stick 12, 11 and 10 of Arduino dormant low mode. When we press set, warning set mode initiates, and now we have to set the alert by utilizing the INC catch, and Next trick will use for moving to the digit. The total breadboard arrangement of this constant clock with a warning.

Program Description

To program for this ongoing clock, we have used a few libraries for extricating time & date from DS1307 and for showing on LCD, which is visible underneath:

```
#include <Wire.h>
#include<EEPROM.h>
#include <RTClib.h>
#include <LiquidCrystal.h>
```

Also, initialization of RTC, LCD, and information output could perform in a setup loop.

```
void setup()
{
 Wire.begin();
 RTC.begin();
 lcd.begin(16,2);
 pinMode(INC, INPUT);
 pinMode(next, INPUT);
 pinMode(set_mad, INPUT);
 pinMode(buzzer, OUTPUT);
 digitalWrite(next, HIGH);
 digitalWrite(set_mad, HIGH);
 digitalWrite(INC, HIGH);
```

Rest of things like reading time, setting the alarm is evident in the void circle section.

```
lcd.print("Time:");
lcd.setCursor(6,0);
lcd.print(HOUR=now.hour(),DEC);
lcd.print(":");
lcd.print(MINUT=now.minute(),DEC);
lcd.print(":");
lcd.print(SECOND=now.second(),DEC);
```

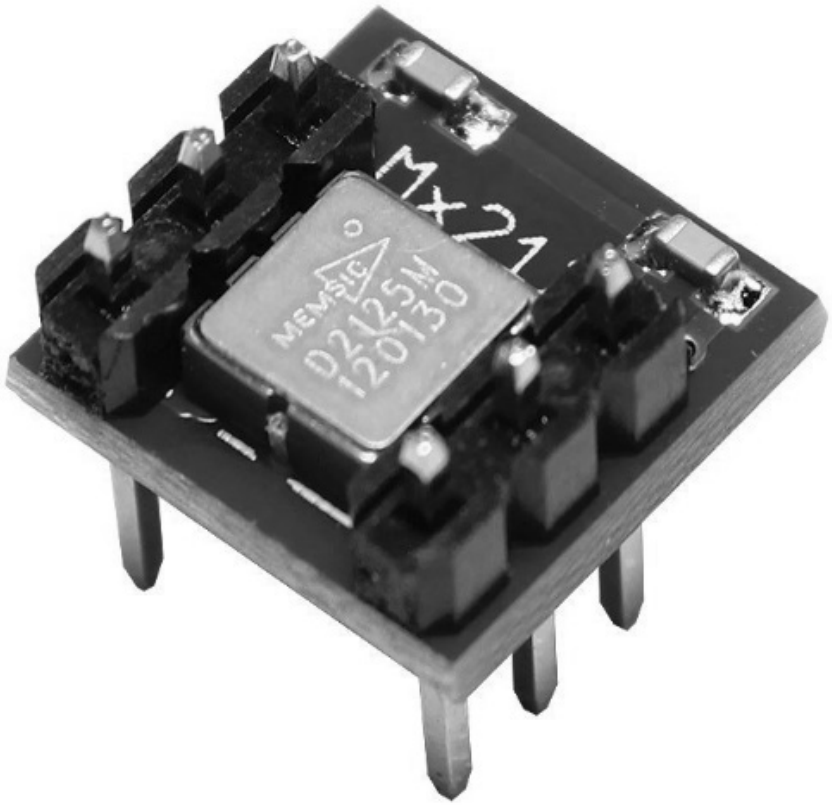
“Stealth and sleuth.”

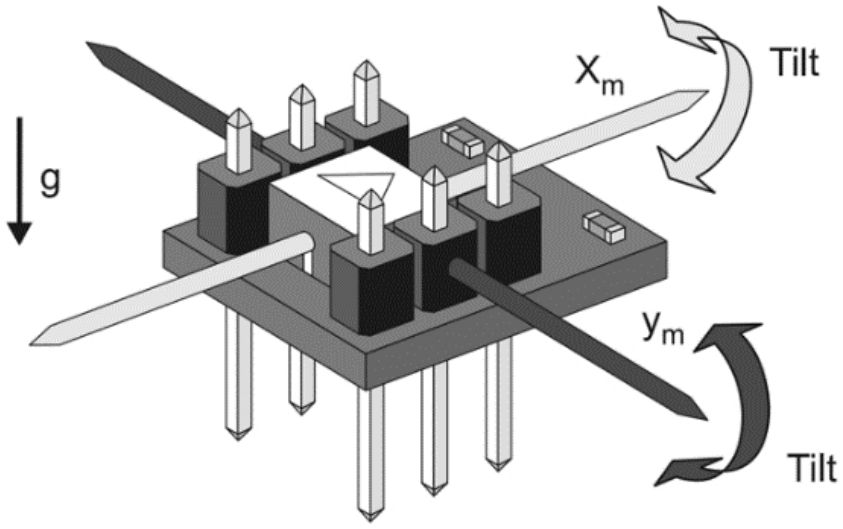
tilt-triggered Coffee Cup Spy Cam camera.

This "James Bond" device could be a murky sword. With the help of an Arduino, this inspection camera can fit into a coffee cup.

The method is to adjust two paper coffee cups — install the device in one, push it into the second, and straighten holes cut in the bottoms of each. Two LEDs should be visible within the regular plastic lid — one illuminates when the tilt switch is activated, the other flashes twice after taking the picture.

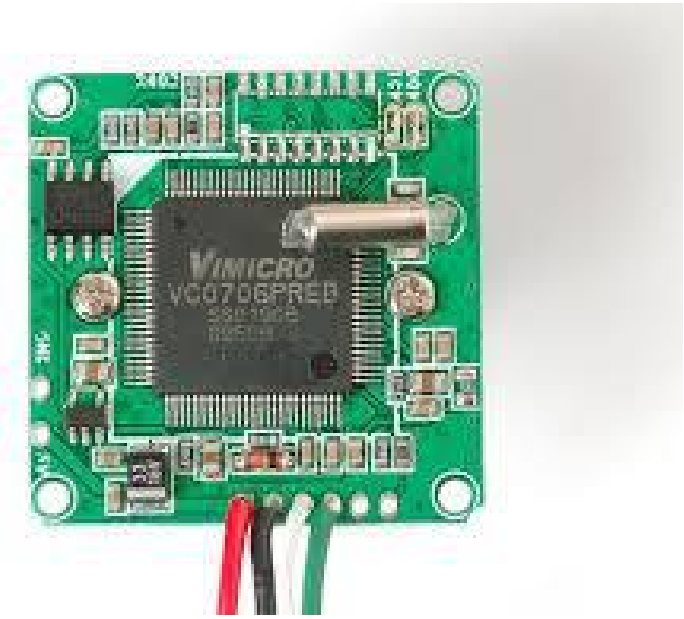
This project connects different modules into a tilt-triggered spy camera that meets inside a coffee cup. The element which triggers the tilt is the Memsic 2125 accelerometer, a dime-sized component which you can see here.





The plan will finish in one paper cup or a plastic bowl with some adjustments to it: the top is cut-off, and a gap is made in the bottom for

the camera's lens to indicate. This cup will then slide into the other bowl/cup, which also should have a hole in its base. With the two holes aligned, when you lift and turn the coffee cup, the LEDs will lighten: one will turn on when the slide switch activates, and the other will flash double after taking the picture.



TTL Serial camera

You will require to test how to fix the camera within the cup. Larger form factor cameras will likely not be able to point straight out the bottom of the bowl/cup, so tinier form factor cameras should be ready to be installed in such a style that the lens spots directly out from spy hole. Don't forget; you need space for the Arduino and SD card, and other parts.

Do an experiment with the TTL Serial camera you get, seize some extra paper coffee cups, and work around with how to install your camera. Check the image below; I have practiced quite a big spy hole, to ensure the lens detects sufficient light to captures a quality image. If you want to decrease the size of the spy hole, use the diameter of your camera's lens, and the size of the cup/s you are using to carry the project.

Chapter 6

C Language Basics and Functions

C programming

C is very famous, straightforward, adaptable programming and a broadly useful programming language. It is machine-autonomous, an organized programming language which is utilized widely in different applications.

C was the nuts and bolts language to compose everything from operating systems to complex projects like the Oracle database, Git, Python translator, and more.

It is common now that 'C' is a divine being's language. So “C” is a base for the programming. If you know 'C,' you can rapidly get a handle on the information from additional programming languages that uses the concept of 'C.'

It is necessary to have extensive experience with PC memory systems since it is a significant perspective when managing the C programming language.

Basics of C language

Here you'll gain proficiency with some fundamental ideas of C Language, to be specific Variables, Tokens, Operators in C language.

How about we begin.

Variables

They are temporary memory locations allotted during the execution of the program. As the name recommends, it is an entity whose worth may change during system execution.

Standards for variable name

a) It doesn't have a space between characters.
b) Utilize a gathering of letters in order, digits, and underscore (_).
c) The principal character ought to be either letter in order or underscore (_).
d) There is no other uncommon image aside from underscore (_).
e) Keywords can't use as a variable name.
f) Revelation of variable

Syntax:

```
datatype variable_name;
```

Example:

```
int a;
```

It tells the client what the data type of an announced variable is or what sort of qualities it can hold.

Instatement of a variable

Here you will see the way toward relegating any an incentive to any variable.

Example:

```
int a = 5;
```

Token

The essential and littlest unit of C program is token**.**

Token incorporates:

1. Keywords

2. Identifier
3. Constants
4. String Constant
5. Operators
6. Special Symbols e.g: _ , @ , *

1. Keywords or Reserve words

These are the words, the compiler knows.

They can not use as a variable name in such a case that we do as such, that implies we are defining the new importance to the compiler which the compiler does not permit.

2. Identifier

They are the name given to programming components, for example, array, function, variable. There are the same standards as of variable name**.**

3. Constant

There are objects whose worth does not exist during the execution of a program.

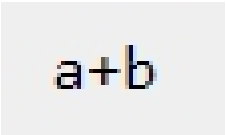
4. String "constant."

Accumulation of character encased by a twofold rearranged comma. e.g., "ABC."

5. Operators

They are used to perform arithmetic and legitimate activities by ALU.

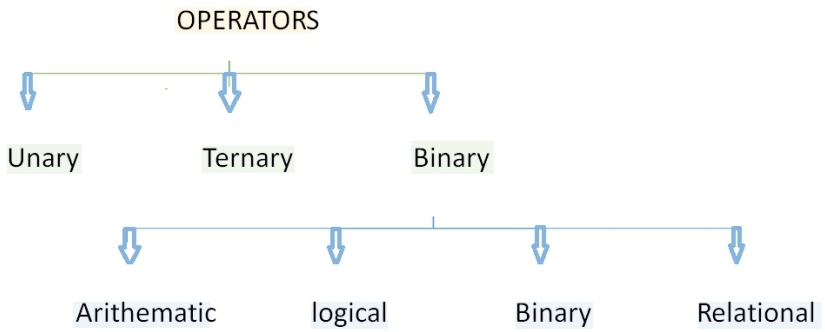
Example:



Here, "**A**" and "**B**" are operands, and "**+**" is an operator.

C language has prosperous operators. A wide range of sorts of operators is accessible in C language for various mathematical countings.

They are, for the most of three sorts:



Operators itself is a different subject which should cover in detail.

Thus, here we begin.

Unary operators

They require just a single operand for execution; it incorporates :

a. Unary less
b. Bitwise compliment
c. Logical Not
d. Increment/Decrement
e. sizeof() operator

sizeof() Operator

It is utilized to restore the size of an operand. It can apply on factor, constant, datatype.

Syntax:

```
sizeof(operand);
```

Example:

```
int a=2, b;
b = sizeof(a);
//or
b = sizeof(int);
//or
b = sizeof(5);
printf("%d\n",b);

//All of the 3 three statements will give same output.
```

Ternary administrators

They are too called contingent operator. For these operators, we require three operands for execution. There is just and one ternary administrator in C language.

Syntax:

```
expression_1  ? expression_2 : expression_3 ;
```

If

`expression_1` is true `expression_2` gets executed, if false then `expression_3`.

Example:

```
int a=4 , b=7 , c;

c = ( a<7 ? a:b );

printf("%d\n", c );
```

OutPut: “4”

Because, a = 4,

exp_1 is true and thus

exp_2 gets executed.

So, c = a gets executed.

Binary operators

a.Arithmetic operators

Addition +

Subtraction -

Multiplication *

Division /

Modulus %

Modulus administrator gives the leftover portion, and division operator provides result. Every arithmetic operators can use for the whole number, and float values except modulus, which can use for integers only.

b. Relational operators

They will use for a comparison between two values. They return the result as true or false.

Less than <

Less than or equal to <=

Greater than >

Greater than or equal to >=

Equal to ==

Not equal to !=

c. Logical operators

Used to consolidate two relational expressions, and they return the outcome as true or false.

A	B	A && B	A || B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

d. Bitwise administrators

They are used to play out the activity on individual bits. They can connect on char and int.

Operators	Symbol name	Meaning
&	Ampersand	Bitwise AND
|	Pipe	Bitwise OR
^	Caret	Bitwise “X OR”
~	Tilde	Bitwise compliment
<<	Double less than	Left shift operator
>>	Double greater than	Right shift operator

C Operator Precedence Table:

This page records C operators arranged by priority. Their associativity shows in what request operators of equivalent priority in the expression are connected.

Precedence	Operator	Description	Associativity

1	++ -- ( ) [ ] . –> (*type*){*list*}	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer Compound literal(C99)	Left-to-right
2	++ -- + – ! ~ (*type*) * & sizeof _Alignof	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT Type cast Indirection (dereference) Address-of Size-of Alignment requirement(C11)	Right-to-left
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ –	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	= += –= *= /= %= <<= >>= &= ^= |=	Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Functions in C Programming

Every C program has a primary () meaning. It is undoubtedly possible to write a successful program in which the only “function” is primary (). It’s true that in specific simple applications, no other function will require.

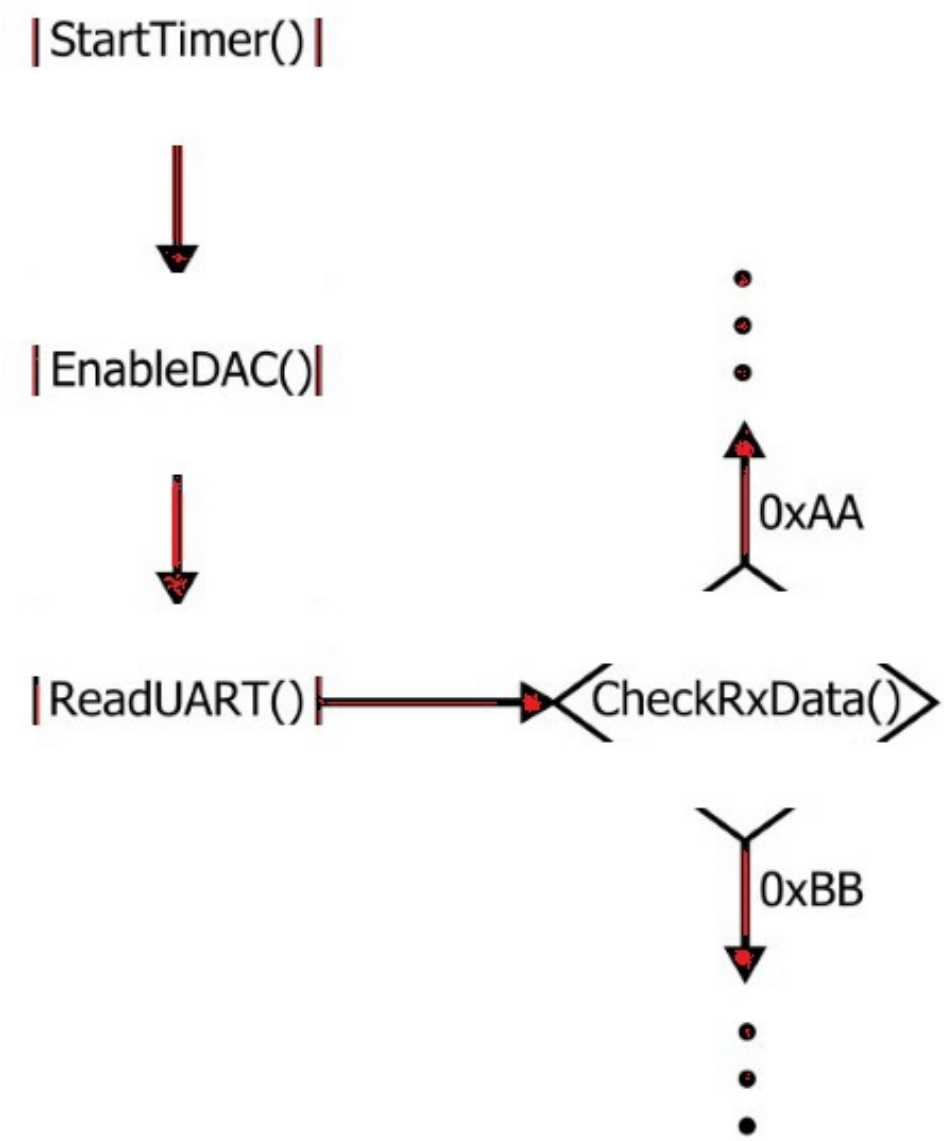
Though, the general use of tasks is a sign that the person inscription the code is a knowledgeable developer. Why? Since "functions" enable us to compose better code all the more rapidly, with less work and fewer bugs. For the individuals who spend a critical bit of their expert life

composing firmware, these are the kind of focal points that can't overlook. Regardless of whether we at first resist the functions of capacities since they appear to require more work, experience step by step instructs us that the advantages significantly exceed the expenses.

Function

"C" function is a collection of directions that work together to perform a particular kind of processor exercise. Much of the time, a "function" will play one explicit undertaking, for example, recovering data from an SPI buffer, designing a clock with the goal that it creates a particularized pause, or reading a value from memory and stacking it into the register of DCA.

Notwithstanding, there is undoubtedly no law expressing that function can perform just one task. You may think that its helpful to have one "function" that updates three random state machines, or you could compose a "function" that transmits a byte by means of UART, at that point checks the status bit until a byte took, at that point value the estimation of the got byte into some geometrical



Components of a Function

A function consists of a name, a rundown of info parameters, the code statements that execute the required functionality, and a return type.

The accompanying code piece gives you an example.

```
char Convert_to_Lowercase(char UppercaseLetter)
{
     if(UppercaseLetter < 65 || UppercaseLetter > 90)
          return 0x00;
     else
          return (UppercaseLetter + 32);
}
```

I like to make my function names exceptionally clear. It makes the code progressively decipherable and causes you to keep your ideas composed.

The guidelines include in curved brackets; this segment of the function definition is visible as the body of the function. The "return" keyword is utilized to leave the "function" and to recognize which information ought to convey to the already executing portion of code. The return type, set before the function name, distinguishes the data type of the information that will return. It is splendidly adequate to have a capacity that plays out an errand, with no compelling reason to recuperate information. For this situation, you would utilize the keyword “void” rather than a data type.

Passing Data to a Function

The input parameters, additionally called arguments, are encased in brackets and set after the function name. A C function can have different contentions, in which case commas will isolate them. An information type must join every dispute.

In inserted applications, it is frequently not essential to utilize contentions. I can consider two purposes behind this.

In the first place, implanted firmware every now and again interfaces straightforwardly with the device’s hardware, and subsequently, a function can get the data it needs from configuration registers, correspondence registers, or port pins.

Second, straightforward C programs composed for microcontrollers can utilize common factors, i.e., elements that are available all through the program and can be gotten to by any capacity. As I understand it, the

utilization of common factors in application writing computer programs is debilitated, or perhaps "censured" would be the fitting word. In any case, as I would see it, numerous firmware ventures, particularly those entirely composed by one software engineer, can profit by the effortlessness of global variables.

Tune up

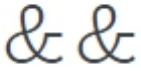
How about we quickly look at the function definition appeared.

- The function name, Convert_to_Lowercase, plainly shows the reason for the function: it recognizes an eight-bit value relating to a capitalized ASCII letter, and it restores an eight-bit value comparing to the lowercase adaptation of that equivalent letter.
- There is one info parameter. It has an information kind of char and uses an explicit identifier.
- The arrival esteem, similar to the information contention, is an ASCII character, and thus the return type is char.
- In the event, that the input value is outside of the range comparing to uppercase ASCII letters, the capacity returns 0x00, which symbolizes an error. Else, it adds 32 to the input value and returns the sum. In case you're curious about ASCII values, the table appeared beneath will assist you with understanding the use of figures given below.
- 65-90-32

Character	Decimal Value	Character	Decimal Value
A	65	a	97
B	66	b	98
C	67	c	99
D	68	d	100
E	69	e	101
F	70	f	102
G	71	g	103
H	72	h	104
I	73	i	105
J	74	j	106
K	75	k	107
L	76	l	108
M	77	m	109
N	78	n	110
O	79	o	111
P	80	p	112
Q	81	q	113
R	82	r	114
S	83	s	115
T	84	t	116
U	85	u	117
V	86	v	118
W	87	w	119
X	88	x	120
Y	89	y	121
Z	90	z	122

Chapter 7

Arduino Logic Statement



Details:

Logical AND results in "true" just if the two operands are "true."

Sample Code:

This operator can be utilized inside the state of an if statement.

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH) { // if BOTH the switches read HIGH
  // statements
}
```

Reminders and Alerts

Ensure you don't mess up the boolean AND administrator, && for the bitwise AND operator and (single ampersand). They are altogether various brutes.

Arduino IDE:

Contingent "if-else-if" Statements

Learn about contingent Statements, and their various sorts in Arduino IDE, for example, the if statement, if-else statement, and if-else-if statement?

IfStatements | Arduino 1.8.5

File Edit Sketch Tools Help

IfStatements

```
int Num1 = 10;
int Num2 = 15;

if(Num1 > Num2){
  Serial.println("Num1 is greater than Num2");
}
else{
  Serial.println("Num2 is greater than Num1");
}
```

Needed Parts

1. EVIVe Prototyping Device

2. USB A-B Cable

Intro:

"Conditional statements" check whether a developer determined Boolean condition is valid or false. They make it conceivable to test a variable against the compare or value a variable and another variable and make the program demonstration in one manner if the condition meets, and different on the off chance that it isn't. They make the program exceptionally incredible and have the option to be utilized for a considerable type of purposes.

This instructional exercise examines the accompanying conditional statements:

1. if statement
2. if-else statement
3. if-else-if statement.

If Statement: Given underneath is the formation of an if statement:

```
If (conditional expression) {
Body of the if statement
}
```

The Conditional Expression can anything which can result either in right or false. On the off chance that the "statement" is valid, the code in

the body of the statement executes. Be that as it may, if the expression turns out as false, the “code” in the "body" removed.

The following is a model appearing at utilizing the if proclamation:

Remember: To program your Arduino from your browser, install the “codebender app” or Arduino Create Agent for “codebender.“

if Statement by STEMpedia | Edit | Clone & Edit | Download

```
int Student1_Marks = 92;

int Student2_Marks = 20;

void setup() {

Serial.begin(9600);

if (Student1_Marks > 33){

Serial.println("The student1 marks is greater than 33. He passed the exam.");

}

if (Student1_Marks <= 33){

Serial.println("The student1 marks is less than or eqaul to 33. He failed the exam."); } if (Student2_Marks > 33){

Serial.println("The student2 marks is greater than 33. He passed the exam.");

}

if (Student2_Marks <= 33){

Serial.println("The student2 marks is less than or eqaul to 33. He failed the exam.");

}

}

void loop() {

}
```

Check the result on the monitor:

- The student1 results are more prominent than 33. He passed the test.
- The student2 results are not exactly or equivalent to 33. The student bombed the test.

In the code above, Student1 has marked more noteworthy than 33; therefore, the central statement is true and executes. For Student2, the subsequent statement is true; consequently, the following statement executes.

if-else statement:

As soon as you use if “statement,” the code in its body runs just when the, if "statement" decides to correct if it decides to false, program execution, hops the given "codes" in the "body" of the if statement and goes to statement the body of the if articulation.

```
If (conditional expression)  {
Body of the if statement when conditional expression is true
}
else {
Body of the else statement when conditional expression is false
}
```

At the point when the conditional expression evaluates to true:

```
Code in the body of the if statement runs.
Code in the body of the else statement does not run.
```

In the end, when the conditional expression evaluates to false:

```
Code in the body of the if statement does not run.
Code in the body of the else statement runs.
```

Remember: To program your Arduino from your browser, install the “codebender app” or Arduino Create Agent for “codebender.“

if Statement by STEMpedia | Edit | Clone & Edit | Download

```
int Student1_Marks = 92;

int Student2_Marks = 20;

void setup() {
```

```
Serial.begin(9600);

if (Student1_Marks > 33){

Serial.println("The student1 marks is greater than 33. He passed the exam.");

}

else{

Serial.println("The student1 marks is less than or eqaul to 33. He failed the exam.");

}

if (Student2_Marks > 33){

Serial.println("The student2 marks is greater than 33. He passed the exam.");

}

else{

Serial.println("The student2 marks is less than or eqaul to 33. He failed the exam.");

}

}

void loop() {

}
```

Check the result on the monitor:

- The student1 results are more prominent than 33. He passed the test.
- The student2 results are not exactly or equivalent to 33. The student bombed the test.

The if-else-if statement

The if-else-if statement enables more than one conditional expressions to evaluates than the if-else “statement.”

The following is the fundamental structure:

```
if (conditional expression 1)  {
Body of the if statement when conditional expression 1 is true
}
else if (conditional expression 2)  {
Body of the else-if statement when conditional expression 1 is false and conditional expression 2 is true
}
else {
Body of the else statement when conditional expression 1 and 2 are both false
}
```

At the point when conditional expression 1 assesses to true:

```
Code in the body of the first if statement runs.
Codes in the body of the else-if statement and else statement don't run.
```

In the end, when conditional expression 1 assesses to false and conditional expression, two evaluates to true:

```
Code in the body of the else-if statement runs.
Codes in the body of the if statement and else statement don't run.
```

At the point when both conditional expression 1 and 2 evaluate to false:

```
Code in the body of the else statement runs.
Codes in the body of the if statement and if-else statement don't run.
```

The following is a model showing to utilize the if-else-if statement:

Remember: To program your Arduino from your browser, install the "codebender app" or Arduino Create Agent for "codebender."

if Statement by STEMpedia | Edit | Clone & Edit | Download

```
int Marks = 95;

char Grade;

void setup() {

Serial.begin(9600);

if (Marks <= 33){
```

```
Grade = 'E';
}
else if (Marks <= 40){
Grade = 'D';
}
else if (Marks <= 60){
Grade = 'C';
}
else if (Marks <= 80){
Grade = 'B';
}
else if (Marks <= 100){
Grade = 'A';
}
Serial.print("Your grade is: ");
Serial.println(Grade);
}
void loop() {
}
```

Check the result on the monitor:

Your grade is A:

Chapter 8

Arduino *for* Loops

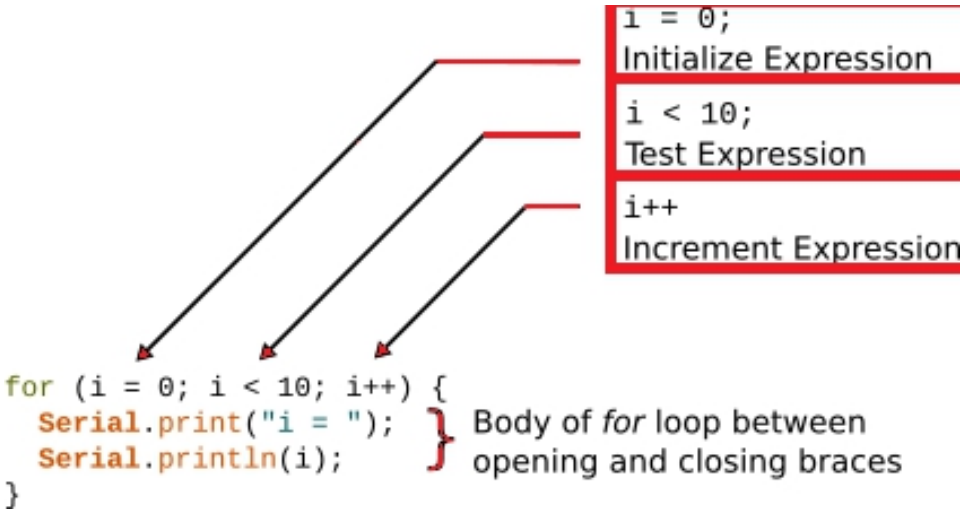
In this chapter, we discuss "for" loop. While observations or code in the Arduino basic circle will run ceaselessly and never leave the "loop," the for loop enables us to loop through "code" a specific number of times before leaving the loop on our instructions.

A typical method to utilize the for loop is with the augmentation operator that covering in the last section of this course.

Using the *for* Loops

```
void setup() {
  int I;
  Serial.begin(9600);
  for (i = 0; i < 10; i++) {
    Serial.print("i = ");
    Serial.println(i);
  }
}
void loop() {
}
```

How the *for* Loop Works



The figure above demonstrates the parts of a loop.

A " *for* " Loop Formation

An essential for loop takes to start as pursues:

Three expressions added between the opening and closing parentheses **()** that determine how many times the statements in the loop will execute before quitting the “loop.” At the point when the “loop” is departed, program execution proceeds underneath the "loop," for example, statements outside and beneath the loop are performed from beginning to end. Loop's body between the opening and shutting brackets {} includes comments that will keep running on top of the "loop." The phrases between the brackets are known as the instate expression, analysis expression, and increment expression. A variable must be characterized to use in the three loop expressions. In the model sketch, a number variable called, I used to use it.

The three loop expressions must show up in the order: begin, test, and increase. Here separate them by semicolons. The addition expression does not finish with (;).

Introduce Expression: The introduce articulation is just run once at the time that the loop begins. It adds our I variable to zero (0) in the model sketch.

Test Expression: The test articulation decides when we break unaware of what's going on. It is utilized to set the occasions that the

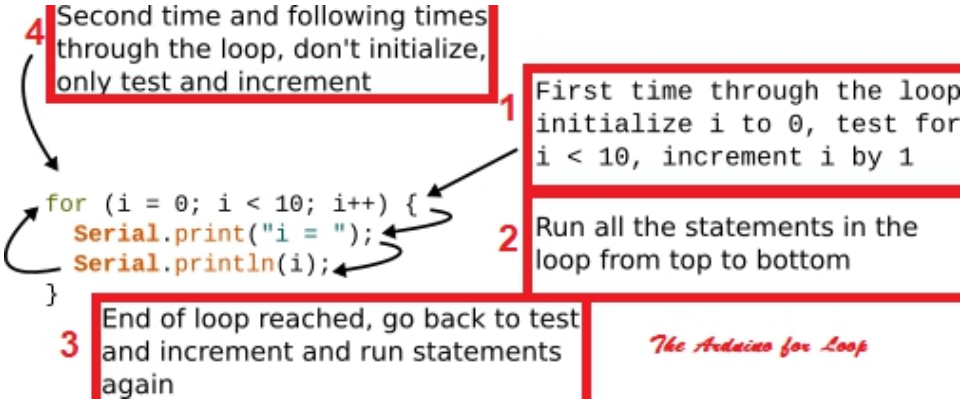
announcements in the body of the loop run.

At the point when the test articulation assesses to genuine, the announcements tuned in the body will run. At the end, when the test expression determines to false, the loop won't run once more, however, will be left.

The test articulation evaluates each time that execution begins at the highest point of the loop.

Augmentation Expression: The increment expression is utilized to change the worth that the I variable holds. It runs each time that execution begins at the highest point of the loop.

Program Flow "Loop"



The figure above shows how program flow works in the for a loop.

Program Flow of *for* Loop

Through the Loop, First Time: The first run through the loop, "I" introduces to 0, the test expression tests either "I" < 10 (0 < 10) which is valid, so the statements on the up and up will loop.

Since the post-increment operator uses with the variable, I might augment toward the finish of the loop. The statements on the "loop" run and print the value of "i" as 0 since it has not yet increased.

We, in this manner, have this:

```
i is initialized to 0
i contains 0
i < 10 evaluates to true or 1 because i is less than 10
The two statements in the loop run, print i as 0
At the end of the loop i is incremented so i == 1
```

Through the Loop, Second Time: Through the loop, a second time, "i" presently contain "1" as it increases at the base of the "loop." The test expression currently tests whether "i" < 10 (1 < 10) which is real, so the statements in the loop will execute repeatedly. The "i" variable will augment to 2 at the base of the "loop," so "1" will print to the sequential screen window.

We presently have this:

```
i is not initialized again
i contains 1
i < 10 evaluates to true or 1 because i is less than 10
The two statements in the loop run, print i as 1
At the end of the loop i is incremented so i == 2
```

Through the Loop, Last Time: Execution of the loop will proceed and "i" will increment forever.

The last time through the circle, we have this:

```
i is not initialized again
i contains 9
i < 10 evaluates to true or 1 because i is less than 10
The two statements in the loop run, print i as 9
At the end of the loop i is incremented so i == 10
```

Execution begins at the highest point of the loop once more; the evaluation expression tested again here.

We presently have this:

```
i is not initialized again
i contains 10
i < 10 evaluates to false or 0 because i is not less than 10 (it is equal to 10)
The statements in the loop are not run again
The loop is exited
The statement below the closing bracket of the loop will be run
```

Elective Way of Writing the *for* Loop: The sketch that pursues shows that the variable utilized on the loop expressions can similarly be set on the loop and does not need to be placed outside the "loop" as the earlier sketch ensured.

```
void setup() {
  Serial.begin(9600);

  for (int i = 0; i < 10; i++) {
    Serial.print("i = ");
    Serial.println(i);
  }
}

void loop() {
}
```

When a Loop Within a Loop: The following representation utilizes a for loop inside the Arduino main loop.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  for (int i = 0; i < 10; i++) {
    Serial.print("i = ");
    Serial.println(i);
  }
  delay(1000);
}
```

We understand here, the for “loop” works precisely equivalent to it did previously, however at this point after it exited, the delay() function is hurried to allow a 1-second delay. The finish of the Arduino primary loop loop() reaches to, so the for loop is run once more.

At the point when the for loop run once more, "i" is instated to 0 because the for loop is being begun from the top once more. It at that point runs again as recently portrayed.

Here the for loop and delay() function will be run ceaselessly since the first Arduino loop unless outflows.

Chapter 9

Ardunio Operators

An operator is a figure that advises the compiler to perform explicit mathematical or logical functions. C language is strong in integral operators and gives the following types of operators:

- Arithmetic Operators
- Comparison Operators
- Boolean Operators
- Bitwise Operators
- Compound Operators

Arithmetic: Variable "a" takes "10" and variable "b" takes "20" therefore:

Operator name	Operator simple	Description	Example
assignment operator	=	Stores the value to the right of the equal sign in the variable to the left of the equal sign.	A = B
addition	+	Adds two operands	A + B will give 30
subtraction	-	Subtracts second operand from the first	A - B will give -10
multiplication	*	Multiply both operands	A * B will give 200
division	/	Divide numerator by denominator	B / A will give 2
modulo	%	Modulus Operator and remainder of after an integer division	B % A will give 0

Boolean: Variable "a" takes "10" and variable "b" takes "20" therefore:

Operator name	Operator simple	Description	Example
equal to	==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true
not equal to	!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true
less than	<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true
greater than	>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true
less than or equal to	<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true
greater than or equal to	>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true

Comparison: Variable "a" takes "10" and variable "b" takes "20" therefore:

Operator name	Operator simple	Description	Example
and	&&	Called Logical AND operator. If both the operands are non-zero then then condition becomes true.	(A && B) is true
or	||	Called Logical OR Operator. If any of the two operands is non-zero then then condition becomes true.	(A || B) is true
not	!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false

Bitwise: Variable "a" takes "60" and variable "b" takes "13" therefore:

Operator name	Operator simple	Description	Example
and	&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
or	|	Binary OR Operator copies a bit if it exists in either operand	(A | B) will give 61 which is 0011 1101
xor	^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
not	~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
shift left	<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
shift right	>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Compounds: Variable "a" takes "10" and variable "b" takes "20" therefore:

Operator name	Operator simple	Description	Example
increment	++	Increment operator, increases integer value by one	A++ will give 11
decrement	--	Decrement operator, decreases integer value by one	A-- will give 9
compound addition	+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand	B += A is equivalent to B = B+ A
compound subtraction	-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand	B -= A is equivalent to B = B - A
compound multiplication	*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand	B*= A is equivalent to B = B* A
compound division	/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand	B /= A is equivalent to B = B / A
compound modulo	%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand	B %= A is equivalent to B = B % A
compound bitwise or	|=	bitwise inclusive OR and assignment operator	A |= 2 is same as A = A | 2
compound bitwise and	&=	Bitwise AND assignment operator	A &= 2 is same as A = A & 2

Arduino Conditional Operator

The conditional operator is different decision-making put together in Arduino programming.

The conditional operator comes with a condition, which can assess to true or false, and two expressions.

If the condition assesses to "true," the conditional expression ends up equivalent to the initial phase. If the status determines to false, the "expression" ends up equivalent to the 2nd expression.

The remaining of this portion of the Arduino programming course will clarify and show how the conditional operator works.

Logical Operators

These operators can be practiced with if and if-else to rearrange and expand decision making.

OR (||), AND (&&) and NOT (!) are the three operators who clarified and showed in chapter seven.

Chapter 10

Decision Making

We make decisions in the Arduino programming language with if statement. The “if “statement” will examine if a form is "true" and if so will execute the block of code inside the brackets.

See the example below, the syntax for the *if* statement:

```
if (condition) {
  // Code to execute
}
```

We can utilize an *else* “statement” after the *if* statement to execute a block of code if the condition isn't "true."

See the example below, the “syntax” for the *if/else* statement:

```
if (condition) {
  // Code to execute if condition is true
} else {
  // Code to execute if condition is false
}
```

The condition, in the, *if* “ statement,” can be any Boolean worth or activity that returns a Boolean outcome. You will find that most of the *if* “statement” in your code will contain correlation tasks. How about we see some “code” that will outline this:

```
if (varA > varB) {
  Serial.println("varA is greater than varB");
} else {
  Serial.println("varB is greater or equal to varA");
}
```

Chapter 11

Arduino Sensors, Input, & Output

Arduino Sensors

As you know already, Arduino is an easy to use, inexpensive, versatile, and powerful tool that can be used to make some very, sensors.

I'll try to explore the basics of getting started with Arduino and provide you with some necessary code that will work with many sensors.

If you need your Arduino to sense it's general surroundings, you should include a sensor. There inexpensive a wide range of sensors to choose from, and they each have a specific purpose.

There are two main categories, known as:

1. Analog

Analog sensors differ extensively in function; however, most work under a similar primary principle. They are, for the most part, resistors that will change their resistance in light of improvement. At the point when the resistance changes, they expend an alternate measure of voltage. If we know how the energy relates to the stimulus, we can easily use the voltage output to determine the value of whatever it is we are trying to measure.

2. Digital

These sensors will, in general, be less complicated in function than analog sensors. You will feel analog sensors give a variable voltage that

changes based on whatever they are measuring. Digital sensors go about as breakers. They are unless on or off and will possibly turn state when an edge signal reaches. There are exceptional cases to this, yet these sensors will, in general, be increasingly confounded and won't be secured here. Try to attempt our digital temperature sensor manage, or the Linear Hall attractive module control, if you already one of the sensors.

Inferior to, you will discover a portion of the commonly used sensors in ventures.

- Distance Ranging Sensor
- PIR Motion Sensor
- Light Sensor
- Degree of Flex Sensor
- Pressure Sensor
- Proximity Sensor
- Acceleration Sensor
- Sound Detecting Sensor
- RGB and Gesture Sensor
- Humidity and Temperature Sensor

The most effective method to Use Arduino's Analog and Digital Input/Output (I/O)

This Arduino can "input and output" analog signals just as electronic signals.

An analog sign is one that can take on any number of values, dissimilar to an advanced sign which has just two benefits:

High and Low. To regulate the analog signals, the Arduino has a built-in simple to-advanced converter (ADC). The ADC transforms the analog voltage into a digital worth. The function that you use to acquire the value of a straightforward sign is analogRead(pin). This function changes over the power of the voltage on an analog input pin and returns a digital an incentive from 0 to 1023, concerning the reference value. The repudiate reference voltage is 5 V (for 5 V Arduino sheets) or 3.3 V (for 3.3 V Arduino sheets). This pin is the only parameter.

There is a built-in digital-to-analog converter (DAC) in this Arduino. It can beat width modulate (PWM) a digital sign to accomplish a portion of the functions of an analog output. The "function" worked to output a PWM sign is analogWrite(pin, value). The pin is the PIN utilized for the PWM signal. The "value" is a symbol corresponding to the charge cycle of the sign. At the point when worth = 0, the message is continuously off. At the point when worth = 255, the signal is still on.

On most maximum Arduino sheets, the PWM function is accessible on the pin, the numbers 3, 5, 6, 9, 10, and 11. The PWM signal frequency on most pins is roughly 490 Hz. On the Uno and comparative sheets, pins 5 and 6 have a rate of about 980 Hz. Pins 3 and 11 on the Leonardo additionally keep running at 980 Hz.

To map an analog input value, which ranges from 0 to 1023 to a PWM output signal, which ranges from 0 – 255:

You can use these five parameters, function map. First is the variable, analog valued and other is 0.1023. 0 & 255 separately.

Chapter 12

Computer Interfacing with an Arduino

Thriving, Both the USB and Bluetooth all believer to Serial I/O, which goes to the UART on the chip. The "most effortless" approach to converse with the Arduino is through via a serial port, which needs just the regular system drivers.

There are several clones Arduino's spare intricacy and cost by supporting Serial (RS232), going through a MAX232 chip or comparable. My Adafruit Boarduino is of this sort - I utilize a sequential link to my MAX232 based Serial-to-TTL link, which associates legitimately to the port pins (Rx/Tx) on the BoArduino. That way, the Arduino doesn't have to keep that hardware installed, decreasing capacity and value. Most "regular" Arduino utilizes a chip on the board to change over from USB to Serial, which is then associated a similar way, through a max232 (look on the board) sequential to-TTL transformer.

SO... Sequential the appropriate response. How you GET to subsequent; that is the issue.

I did not use any of the remote arrangements however as I comprehend XBEE, just as most Bluetooth connectors, basically go about as a virtual wire.. the PC and Arduino talk over a sequential connection link via Bluetooth, the main driver you will need is for your very own Bluetooth dongle or inherent on your PC. When matched, neither the laptop nor the Arduino know or care that the connection is remote. I have a more

established inkjet printer that uses Bluetooth as a connection choice. The original drivers to make that printer work are another issue. However, the association is primary. The range for Bluetooth is supposed to be something like 10m; I find around 20 feet more realistic for most applications in a noisy electrical environment when using other Bluetooth-connected items.

Connecting with different software

You have gone to the spot to find out about interfacing an Arduino to different devices, whatever software is operating on those different devices. This Arduino can "talk," "transmit or send and receive data" using a sequential channel, so some other devices with subsequent abilities can speak with an Arduino. It doesn't make a difference what program/programming language is driving the other device. You can unless utilizing the Arduino's "primary" serial port, the one it uses when you "talk" to it to program it, or you can leave that channel devoted to programming, and utilize two different pins for a serial link committed to the external device.

A few projects don't have primary serial inclinations. They can, however, interact with Arduino through an emissary which, similar to an "interpreter," empowers them to converse with one another.

Chapter 13

In-depth Computer Science Topics

There’s never been a brighter outlook for computer science lovers than today. As specific current figures present, computer science learners are in before-mentioned great interest that they can stand to be fussy about the nature of business and trade they are going to select. What's more, it's not hard to perceive any reason why. Innovation has been developing so exponentially over new years; there has been a relentlessly expanding interest for splendid alumni to come in and help to change regions extending from information foundation to cybersecurity.

If you are keen on seeking after a career in software engineering, it's fundamental to keep contemporary with the most recent patterns in software engineering research, to settle on an educated decision about where to head straightaway. Look at these five patterns raging the tech business!

1) Computer-assisted education

2) Cybersecurity

3) Big data analytics

4) Artificial intelligence and robotics

5) Bioinformatics

1. Computer Assisted Education

There is sufficient usage of workstations and software to support learning and coaching; computer-supported knowledge delivers many advantages and has various uses. For learners with learning limitations, for example, it can present personalized guidance and empower students to study at their velocity, delivering the teacher to dedicate more time to each. The area is still developing but engaging, with many instructors admiring its experience to allow students to mesh in active, self-governing, and play-based training.

2. Cybersecurity

As reported by the US BLS (Bureau of Labor Statistics), cybersecurity jobs are prophesied to expand by 28% from 2016 to 2026, much agile than ordinary for all professions, and parenting interests about the deficit in qualified bachelors. In 2-2015, Barack Obama spoke of the need to "collaborate and explore partnerships that will help develop the best ways to bolster our cybersecurity." It's not difficult to explain why he might think so. We live in a hyper-connected world, in which absolutely everything – from banking to dating to governmental infrastructure – is done online. In today's world, data protection is no longer optional, for either individuals or nations, making this another growing strand of computer science research.

3. Big Data Analytics

In early 2012, the Harvard Business Review marked data science the 'most suggestive job' of the new era. There has been a rush in interest for specialists in this area and increased efforts on the part of trademarks and agencies to expand wages and draw data science talents. From investment to health-related, big data analytics is throughout, as companies frequently venture to make more extensive use of the large datasets they have, to personalize and enhance their services.

4. Artificial intelligence and robotics

Among the worldwide robotics trade forecast to be worth around US$80 billion up to 2024, a substantial share of this growth is down to the force of interest and stake in AI, mostly know as artificial

intelligence, one of the most contentious and entertaining areas of computer science research. This technic is still in its initial stages, but tech titans like Facebook, Google, and IBM are spending vast amounts of money and means into AI study. There's no deficiency of chances to develop real-world applications of the technology, and there's an extensive range for break-through bits in this field.

5. Bioinformatics

It is an engaging application of big data, bioinformatics, or the exploitation of programming and software progress to develop large datasets of vital information for research ideas, provides immense latent. While linking huge pharma companies with software companies, bioinformatics is increasing in demand and allows good job hopes for computer science researchers and graduates inspired in biology, medical technology, pharmaceuticals, and computer information science.

Chapter 14

Arduino API Libraries

The Arduino setting can grow within the usage of libraries, only like most programming stages. Libraries present additional functionality for use in drawings, e.g., operating with hardware or handling data. To utilize a library in a sketch, choose it with:

Sketch > Import Library.

Several libraries arrive installed with the IDE. However, you can further download or build your own. You can try the API Style Guide for information on making a good Arduino-style API for your library.

Usual/Standard Libraries

- **EEPROM:** read and write to "constant" area
- **Ethernet:** toward attaching to the internet utilizing this shield
- **Firmata:** during communicating including applications toward the computer practicing the following standard order.
- **GSM:** as connecting to a GSM/GRPS network among that GSM shield.
- **LiquidCrystal:** during controlling LCD's (liquid crystal displays)

- **SD:** as read and write Secure Digital cards
- **Servo:** as testing servo motors
- **SPI:** to interacting by devices utilizing unique SPI (Serial Peripheral Interface)
- **SoftwareSerial:** to a serial interface at each stud.
- Stepper: toward managing stepper machines
- **TFT:** during rendering text, pictures, and sketches at the Arduino TFT shade
- **WiFi:** during attaching to the internet utilizing the Arduino WiFi shield
- **Wire:** Couple of Wire Interface for transmitting and collecting data over a set of tools.
- This **Matrix and Sprite** libraries are no more section of the core configuration.

All Libraries

A list of the *2303* libraries registered in the Arduino Library Manager.

All Libraries see here: https://www.arduinolibraries.info/libraries

Chapter 15

Using the Stream Class & Working with Strings

Using the Stream Class:

Description

A stream is the core class for "character and binary" under streams. It is not described instantly but entreated wherever you practice a function that depends upon that.

A Stream describes the read functions in Arduino. While practicing any center functionality which practices a read() or same scheme, you can securely believe it calls on the Stream section. During functions similar print(), Stream receives from the Print class.

Few of the libraries that depend on Stream comprise :

- Serial
- Wire
- Ethernet
- SD

Working with Strings

Strings can use to stock text. They can use to present "text" on an LED or in the Serial Monitor Window of Arduino IDE.

Strings are too helpful for collecting user input, for example, the characters such a user signs on a keypad attached over the Arduino.

Presently there are a couple of strings in Arduino programming:

A. Arrays of characters that are similar to the "strings" applied in the programming of C.

B. The Arduino String that lets us handle a string intention in a plan.

I'll explain strings, objects and use of these in Arduino sketches in this section. The topic of whoever sort of a pain to use in design to answer at the end of the chapter.

The primary sort of string which we will examine at is the string which is a set of characters of type scorch. An array is a continuing series of the very type of variable saved in memory. A "string" is an array of scorch variants.

A string is a unique array which has a whole additional element at the edge of the "string," that forever has the value of 0. It is famous as a "null-terminated string."

A Sample Sketch of String Character Array

The sketch would show you in what way to obtain a string and print it to the serial monitor window.

```
void setup() {
  char my_str[6];    // an array big enough for a 5 character string

  Serial.begin(9600);

  my_str[0] = 'H';  // the string consists of 5 characters
  my_str[1] = 'e';
  my_str[2] = 'l';
  my_str[3] = 'l';
  my_str[4] = 'o';
  my_str[5] = 0;    // 6th array element is a null terminator

  Serial.println(my_str);
}

void loop() {
}
```

The provided “sketch” explains what a “string” is composed of; it made-up of a character array among printable characters and a 0 in the closing element of the "array" to determine which this is under the string stops.

The string would be print out to the Arduino IDE Serial Monitor window by utilizing Serial.println() and transferring it the name of the "string."

This similar sketch can be composed more helpfully along with these means:

```
void setup() {
  char my_str[] = "Hello";

  Serial.begin(9600);

  Serial.println(my_str);
}

void loop() {
}
```

In the above “sketch,” the compiler determines the dimension of the string array also spontaneously null terminates the string by a zero. An "array," which is six elements, significant and made-up of five characters accompanied by zero created precisely the related way as in the early sketch.

Characters and Strings: Characters can write within single quotes like here:

```
'w'
```

Strings are written between double quotes like this:

```
"This is a string"
```

How to Manipulate String Arrays: One can easily change a string array inside a "sketch" while the following "sketch" shows.

```
void setup() {
  char like[] = "I like coffee and cake";  // create a string

  Serial.begin(9600);

  // (1) print the string
  Serial.println(like);

  // (2) delete part of the string
  like[13] = 0;
  Serial.println(like);

  // (3) substitute a word into the string
  like[13] = ' ';  // replace the null terminator with a space
  like[18] = 't';  // insert the new word
  like[19] = 'e';
  like[20] = 'a';
  like[21] = 0;    // terminate the string
  Serial.println(like);
}

void loop() {
```

Generating and Printing the String: Within the above sketch, a unique string makes and later printed for a show (1).

Reducing the String: The string can short by substituting the fourteenth character in the "string," including a null terminating zero (2).

Here is element number thirteen in the string array numbering of 0.

While the string is printing out, while the characters print up to the new null-terminating zero, the additional characters do not escape – they yet endure in memory, and the string array is still the equivalent size. The sole distinction is that any function that runs with strings will only see the "string" up to the initial null terminator.

How to change a Term in the String: Lastly, the sketch substitutes the term "cake" with "tea" (3).

It original has to substitute the null terminator at like with space so that the string restores to how it created.

New characters squeeze "cak" of the word "cake" beside the word "tea." It is accomplished by overwriting original characters. The 'e' of "cake" restored with a different null terminating character. The outcome is that the string terminates with two "null" characters – the original one at the end of the "string" and the new one that replaces the 'e' in "cake." It made no difference when the original string pointed out because the function that prints the "string" stops printing string characters when it encounters the first null terminator.

Chapter 16

User-defined functions

In this last chapter, you will learn about functions and file handling with Arduino. We learned about loops and conditions already. Let's begin our journey into Functions with Arduino.

What is a Function?

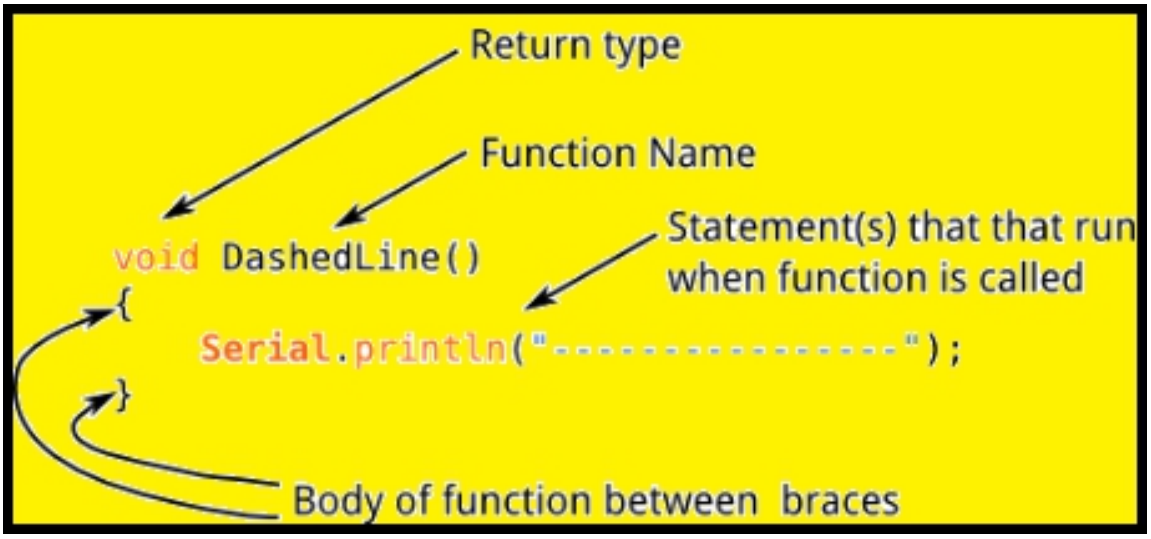
- Each function should have a sole name; setup is a single instance of a unique function name.
- A function name followed by open and close brackets () that may or may not include the object.
- Every function should have a return kind. Set of two setups plus loop have an invalid return sign.
- Here, the body of a function made-up of an open and close bracket ({ and }).

The Structure of a Function

Ahead function can use in a sketch; it must create. The following code is an instance of a "function" that created to print a smashed line in the Arduino IDE.

```
void DashedLine()
{
    Serial.println("----------------");
}
```

The “code” beyond that performs the function is describes as the function definition. The drawing below explains the elements of a "function."



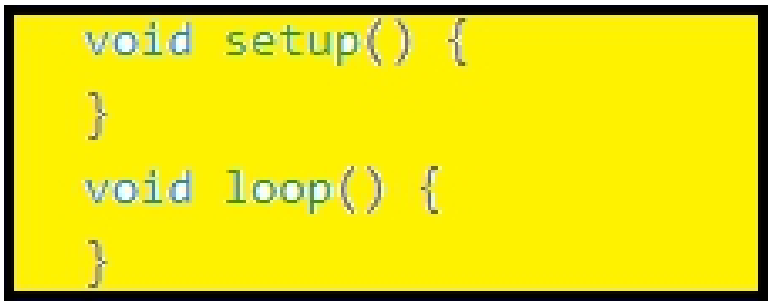
Function Types:

Each function returns sort of, or value, that one is called the return value. The return values can be of different types, a portion of which is here:

- Integers
- Float
- Double
- Character
- Void
- Boolean

The further term, argument, speaks about something that is transferred to the function and used inside the "function." In our earlier instance, the ingredients stated to our "coffee-making" method can say arguments like; sugar, milk, and so on, and we eventually received the coffee, that is the return value of the function.

Through description, there are two types of functions. They are a system-generated function and a user-specified function. In our Arduino code, we have frequently seen the following formation:



setup () and loop (); as well functions. Each return type of certain functions is empty. That's alright, and we will consider the kind of "function" shortly. The *setup () and loop ()* functions are by the system functions. There are several by the system functions. The user-specific "functions" will name after them.

Previously going farther into function types, let’s learn the syntax of a function. Functions can be as follows:

```
void functionName()
{
    //statements
}
Or like
void functionName(arg1, arg2, arg3)
{
    //statements
}
```

Hence, what's the distinction? Sure, the primary “function” has no arguments, but the next "function" seems.

We have four kinds of function, contingent on the return type and arguments. They are as per the following:

- A function with no arguments and no return value
- A function with no arguments and a return value
- A function with arguments and no return value
- A function with arguments and a return value

Presently, the problem is, can the cases be of any sort? Okay, they can be of any kind, contingent on the function. They can be:

- Boolean,
- integers,
- Floats,

Or even characters. They can be a composite of data models too. We will take an example at specific models later. Presently, how about we

characterize and make an example at instances of the four kinds of capacity we portrayed.

You noticed, "functions," including no arguments and no return value, certain "functions" do not allow "arguments." This return type of such functions are void; that determines the "function" returns none. Give me a chance to clear this. We know it already; a "function" must be named. The description of a "function" will follow the command to adjustable naming. If we have a name for a "function," we require to determine its type too. It's the fundamental law for committing a "function." Therefore, if we are not certain our function's class, "what sort of job it will do," then it is secure to use the invalid keyword in front of our "functions," where void indicates no data type, as in the following function:

```
void myFunction(){
//statements
}
```

Within the "function," we may arrange all the stuff we need. Suppose we want to print "I love Arduino"! Ten times if the "function" called. Therefore, our "function" must have a loop which stretches for ten times and later ends. Hence, our function will be as follows:

```
void myFunction() {
  int i;

  for (i = 0; i
```

The earlier "function" does not have a return value. However, if we call the "function" from our primary "function";

From the setup () "function"; we be able to order it from the loop () function, until we do not want an endless loop, the "function" will print "I love Arduino"! Ten times. Regardless of how often we call, it will "print" ten times for every call.

We should compose the full code and take a look at the output. The complete "code" is as per the following:

```
void myFunction() {
  int i;

  for (i = 0; i
```

Use of Function without Arguments & Return Value

In this sort of "function," no arguments are given; however, they return a value. You require to identify that the type of the function influences the return value. If you hold a "function" as an integral function, the return value's type will have to have been a digit too. If you indicate a "function" as a figure, the return type should be a character. It is valid for all other data types so good.

How about we take a look at an example. We will hold an integer function, wherever we will determine a few integers. We will attach them and save them to a different integer, and ultimately, return the extension. The function can look as follows:

```
int addNum() {
  int a = 3, b = 5, c = 6, addition;
  addition = a + b + c;
  return addition;
}
```

The previous function must return 14. Let's save the function's return value to a different integer type of variable in the setup() "function" also print in on the serial monitor.

The entire code would be as follows:

```
void setup() {
  Serial.begin(9600);
  int fromFunction = addNum(); // added values to an integer
  Serial.println(fromFunction); // printed the integer
}

void loop() {

}

int addNum() {
  int a = 3, b = 5, c = 6, addition; //declared some integers
  addition = a + b + c; // added them and stored into another integers
  return addition; // Returned the addition.
}
```

Use of Function without Arguments & No Return Value

This type of function prepares some arguments within the "function," still does not return anything immediately. We can do the computations within the "function" or print something, though there will be no return value.

Suppose we need to figure out the precisely two integers. We may determine the number of variables to store them and later print the sum. Precisely with the guidance of a function, we can move two integers through a "function"; moreover, inside the "function," all we require to do is sum them and save them in a different variable. Later we will print the value. Each time we order the function and move our values within it, we will take the sum of the integers we move. Let's determine a "function" that will show the amount of the two integers passed into the function. We will call the function sumOfTwo(), and because there is no return value, we will determine the "function" as invalid.

The function should look as follows:

```
void sumOfTwo(int a, int b) {
  int sum = a + b;
  Serial.print("The sum is " );
  Serial.println(sum);
}
```

Since we call the "function," including appropriate arguments, the "function" will print the sum of the number we move within the "function."

We should take a look at the output first; at that point, we will talk about the code:

We move the arguments over a function, departing them with commas. The order of the "arguments" does not screw up whereas we call the "function." given the fact that the arguments of a function may be of different types if we spoil it while appealing, the program may not collect and will not perform precisely:

Say a "function" seems as follows:

```
void myInitialAndAge(int age, char initial) {
  Serial.print("My age is ");
  Serial.println(age);
  Serial.print("And my initial is ");
  Serial.print(initial);
}
```

Forthwith, we need to call the function like so: *myInitialAndAge(6, 'T');*,

then 6 is my age and T is my initial. We should not do it as follows:

myInitialAndAge('T', 6);.

As we requested the function and moved two values within it, "12 and 24." We received the output as "The Sum is 36."

We should go somewhat more profound. Within our function, we held our two arguments, "a and b" as integers. Within the entire "function," the values, "12 and 24." we passed through the function are as follows:

```
a = 12 and b =24;
```

As long as we called the “function” here:

sumOfTwo(24, 12),

the values of the variables may be as follows:

```
a = 24 and b = 12;
```

I believe you can instantly recognize the order of arguments of a function.

Call the sumOfTwo()

function five times in the setup() function, including various values of *"a and b,"* and match the outputs.

Use of Function with Arguments & A Return Value

You will notice, this type of function will have one and the other the arguments and the return value. Within the "function," hither will be any processing or calculations utilizing the "arguments," and afterward, there could be an upshot, that one we want as a return value. Because this kind of function will return a value, the "function" gotta have a type.

We should look at an example.

Here we will write a function which will verify if a number is superior or not. Out of your math class, you may be able to retain that a preferred number is a real number more significant than one that has no real splitters other than 1 and oneself.

The primary philosophy following indicating whichever a number is best or not is to check all the names starting from 2 to the number earlier the number alone by cutting the amount.

We should verify if 9 is the top number. No, it is not a high number. Since it can be split by 3, and as reported by definition, the best name cannot divide by any number other than 1 and the number itself.

Therefore, we will check if "9" is separated by 2. No, it is not. Next, we will divide by 3, and yes, it is divisible. So, 9 is not a prime number, as stated in our logic. We should see if

13 is a prime number.
We will check if the number is divisible by 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12. No, the number is not divisible by any of those numbers.

We might likewise reduce our verifying by only checking the "number" that is the share of the "number" we are testing.

Take a look at the following code:

```
int primeChecker(int n) // n is our number which will be checked
{
  int i; // Driver variable for the loop

  for (i = 2; i
```

This code is pretty mild. If the remainder is similar to zero, our number is entirely divisible by a "number" other than 1 and oneself; therefore, definitely, no it's not the main number. If you would notice any remainder, the "number" is a prime number. We should compose the full code and look at the output for the following numbers:

23 65 235 4,543 4,241

The complete source code to verify if the numbers are prime or not is as follows:

```
void setup() {
  Serial.begin(9600);
  primeChecker(23); // We called our function passing our number to test.
  primeChecker(65);
  primeChecker(235);
  primeChecker(4543);
  primeChecker(4241);
}

void loop() {
}

int primeChecker(int n)
{
  int i; //driver variable for the loop

  for (i = 2; i
```

Did you notice this is a straightforward code? We just named our *primeChecker()* function and passed given numbers. Within our *primeChecker()* "function," we signed the philosophy to check our number.

We can recognize from the output, different than 23 and 4,241, none of the numbers are superior.

We should see the example below, under we will write four functions:

- add()
- sub()
- mul()
- and divi().

Within these functions, we will transfer binary numbers and print the value on the serial monitor. The four functions can be defined as follows:

```
float sum(float a, float b) {
  float sum = a + b;
  return sum;
}

float sub(float a, float b) {
  float sub = a - b;
  return sub;
}
float mul(float a, float b) {
  float mul = a * b;
  return mul;
}
float divi(float a, float b) {
  float divi = a / b;
  return divi;
}
```

Use of Functions

You perhaps wonder which type of “function” we should utilize

The solution is naive. The ways of the “functions” are reliant on the operations of the programs. No matter what the "function" is, I would recommend making only a particular task with a "function." So do not do try multiple "tasks" within a function. It will generally accelerate your processing rate and the computation time of the code.

You perhaps also need to understand why we still need to use functions. Yeah, their various methods of “functions,” as regards:

- “Functions” assist programmers compose more standardized code
- “Functions” assist in reducing errors by explaining the encode
- “Functions” perform the entire “code” smaller
- “Functions” build an occasion to utilize the code many spells

Workout

To increase your awareness of “Functions,” you may require to do the following assignment:

- Compose a program to verify if a number is equal or varied.
- Compose a "function" that will obtain the most significant number between the four numbers. The characters will pass as arguments into the “Functions.”
- Assume you operate in a garment factory. They want to understand the area of the cloth. Space can be on the platform. They will give you the length and extent of the fabric. At the moment, write a program managing functions to figure out the area of the structure. Try to use the first calculation in the user-specified "function."

Conclusion

It seems the end of this informative ebook, but it's just the origin of your journey working with Arduino.

Hopefully, now you will have a working sensor. Worse comes to worst I hope you have a slightly better idea of how Arduino works. It is a little bit confusing at first, and it can be annoying, but keep with it. It makes it all the more pleasant when you do eventually figure it out.

This ebook is typically for the primary audience of the Arduino. Though you might have some programming experience that you want to take to the Arduino platform, we will feel no prior knowledge of writing code. With that said, a keen awareness of the computer is crucial, as is the compliance and analytical interest to look beyond this book for specific answers. The preponderance of Arduino users want to get things accomplished and often don't care about the small details, and they want their projects to go. I understand this; at least it wasn't before the Arduino came along. Besides, I was never one for a love of mathematics, which thankfully is not a prerequisite to genuinely enjoy the process of writing code.