LABORATORY MANUAL FOR Internet of Things

6TH Semester

Diploma in Computer Science & Engineering

Department of Computer Science & Engineering

C. V. Raman Polytechnic

Bidya Nagar, Mahura, Janla, Bhubaneswar

(Affiliated to SCTE&VT: Approved by AICTE)

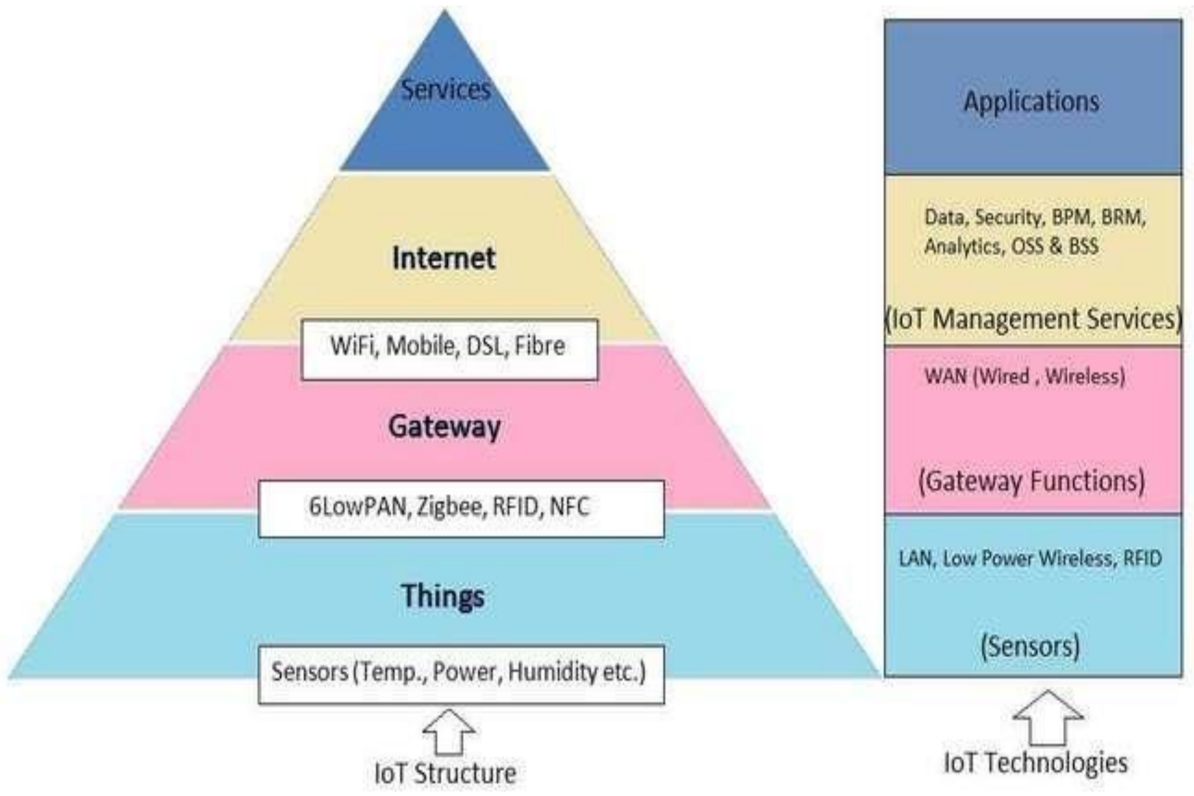
Internet of Things (IoT)

<u>Introduction:</u> IOT stands for “Internet of Things”. The IOT is a name for the vast collection of “things” that are being networked together in the home and workplace (up to 20 billion by 2020 according to Gardner, a technology consulting firm).

<u>Characteristics of the IoT</u>

Networking	These IoT devices talk to one another (M2M communication) or to servers located in the local network or on the Internet. Being on the network allows the device the common abilityto consume and produce data.
Sensing	IoT devices sense something about theirenvironment.
Actuators	IoT devices that do something. Lock doors,beep, turn lights on, or turn the TV on.

Communications in IoT


Services
Internet
WiFi, Mobile, DSL, Fibre
Gateway
6LowPAN, Zigbee, RFID, NFC
Things
Sensors (Temp., Power, Humidity etc.)
IoT Structure
Applications
Data, Security, BPM, BRM, Analytics, OSS & BSS
(IoT Management Services)
WAN (Wired , Wireless)
(Gateway Functions)
LAN, Low Power Wireless, RFID
(Sensors)
IoT Technologies

Communications are important to IOT projects. In fact, communications are core to the whole genre. There is a trade-off for IOT devices. The more complex the protocols and higher the data rates, the more powerful processor needed and the more electrical power the IOT device will consume.

TCP/IP base communications (think web servers; HTTP-based commutation(like REST servers); streams of data; UDP) provide the most flexibility and functionality at a cost of processor and electrical power.

Low-power Bluetooth and Zigbee types of connections allow much lower power for connections with the corresponding decrease in bandwidth and

functionality. IOT projects can be all over the map with requirements forcommunication flexibility and data bandwidth requirements.

Arduino in IoT

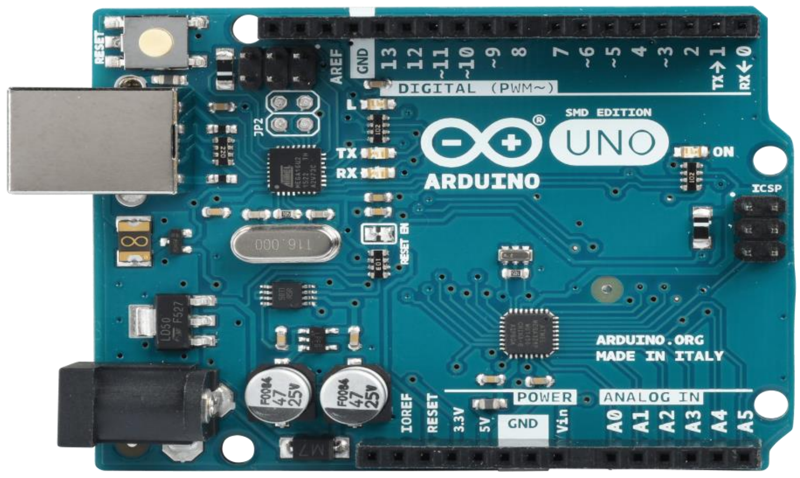
In IoT applications the Arduino is used to collect the data from the sensors/devices to send it to the internet and receives data for purpose of control of actuators.

Arduino Uno

<u>Introduction:</u> The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable. It can be powered by theUSB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts. The word "uno" means "one" in Italian and was chosen to markthe initial release of Arduino Software.

<u>Features of the Arduino</u>

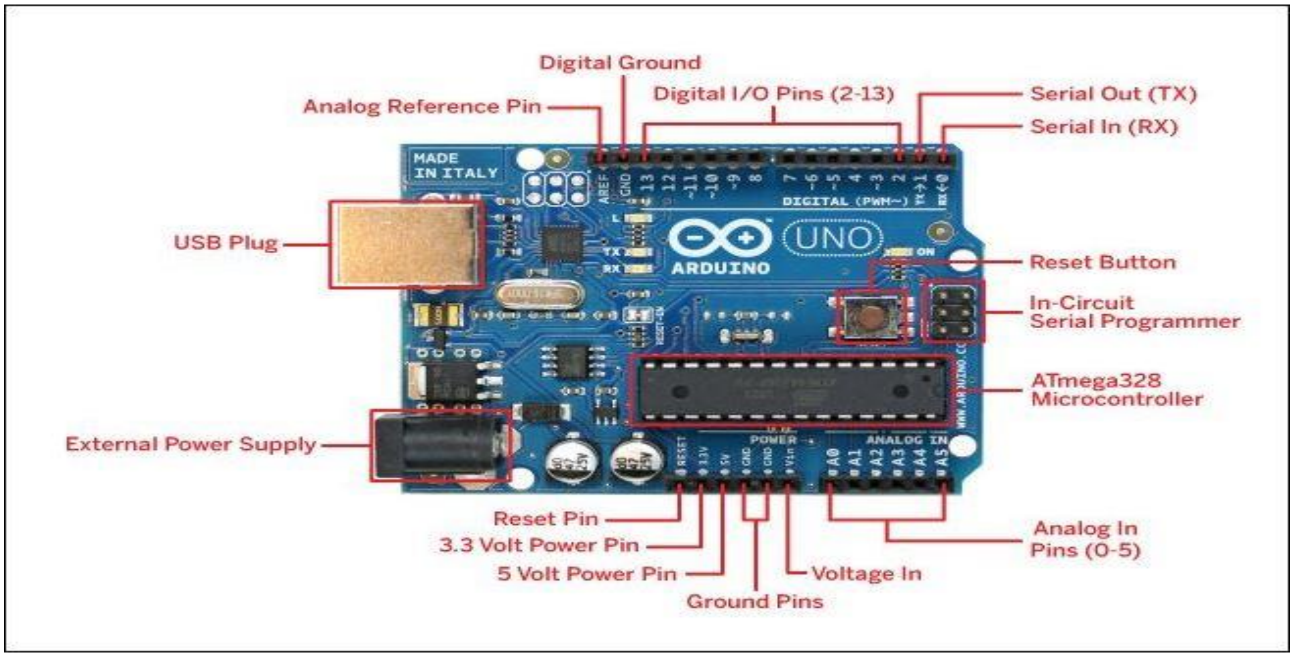
1. Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.



2. The board functions can be controlled by sending a set of instructions to themicrocontroller on the board via Arduino IDE.
3. Arduino IDE uses a simplified version of C++, making it easier to learn toprogram.
4. Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Arduino UNO Pin Configuration

Arduino boards senses the environment by receiving inputs from many sensors, and affects their surroundings by controlling lights, motors, and other actuators. Arduino boards are the microcontroller development platform that will be at the heart of your projects. When making something you will be building the circuits and interfaces for interaction, and telling the microcontroller how to interface with other components. Here the anatomy of Arduino UNO.



1. **Digital pins** Use these pins with digitalRead(), digitalWrite(), and analogWrite(). analogWrite() works only on the pins with the PWM symbol.
2. **Pin 13 LED** The only actuator built-in to your board. Besides being a handy target for your first blink sketch, this LED is very useful for debugging.
3. **Power LED** Indicates that your Arduino is receiving power. Useful for debugging.
4. **ATmega** microcontroller The heart of your board.
5. **Analog in** Use these pins with analogRead().
6. **GND and 5V pins** Use these pins to provide +5V power and ground to your circuits.
7. **Power connector** This is how you power your Arduino when it's not plugged into a USB port for power. Can accept voltages between 7-12V.
8. **TX and RX LEDs** These LEDs indicate communication between your Arduino and your computer. Expect them to flicker rapidly during sketch upload as well as during serial communication. Useful for debugging.
9. **USB port** Used for powering your Arduino UNO, uploading your sketches to your Arduino, and for communicating with your Arduino sketch (via Serial. println() etc.).
10. **Reset button** Resets the ATmega microcontroller.

Arduino IDE
(Integrated DevelopmentEnvironment)

<u>Introduction:</u> The Arduino Software (IDE) is easy-to-use and is based on the Processing programming environment. The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++.The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduinoboard.

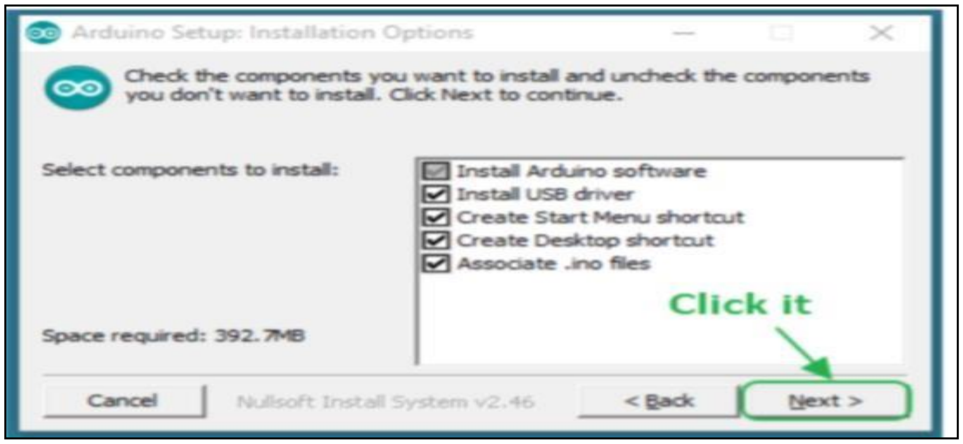
The Arduino Software (IDE) – contains:

- A text editor for writing code
- A message area
- A text consoles
- A toolbar with buttons for common functions and a series of menus.It connects to the Arduino hardware to upload programs and communicate with them.

Installation of Arduino Software (IDE)

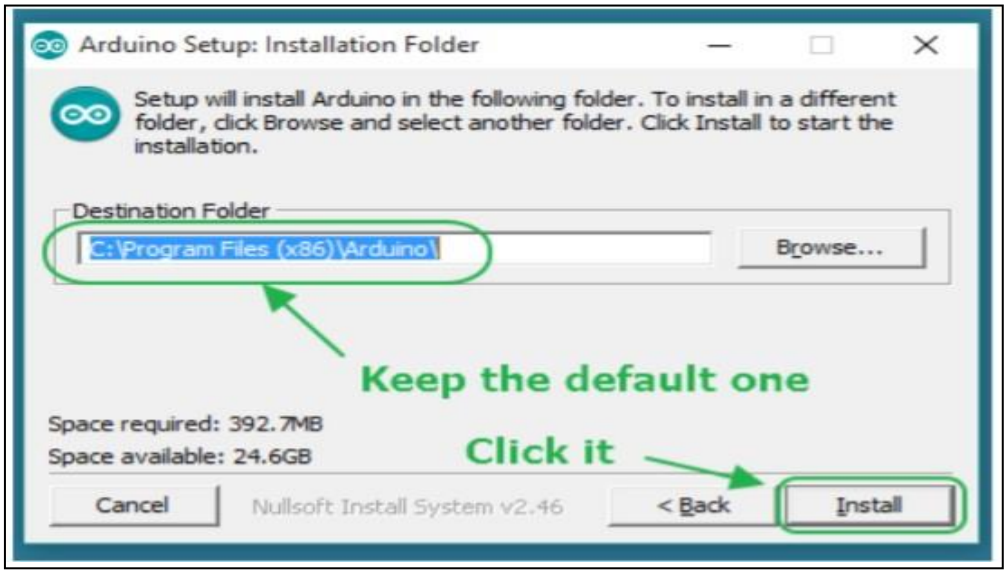
<u>Step1: Downloading</u>

➢ To install the Arduino software, download this page: http://arduino.cc/en/Main/Software and proceed with the installation by allowing the driver installation process.


Arduino Setup: Installation Options
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
Install Arduino software
Install USB driver
Create Start Menu shortcut
Create Desktop shortcut
Associate .ino files
Click it
Space required: 392.7MB
Cancel
Nullsoft Install System v2.46
< Back
Next >

Step 2: Directory Installation

Choose the installation directory.


Arduino Setup: Installation Folder

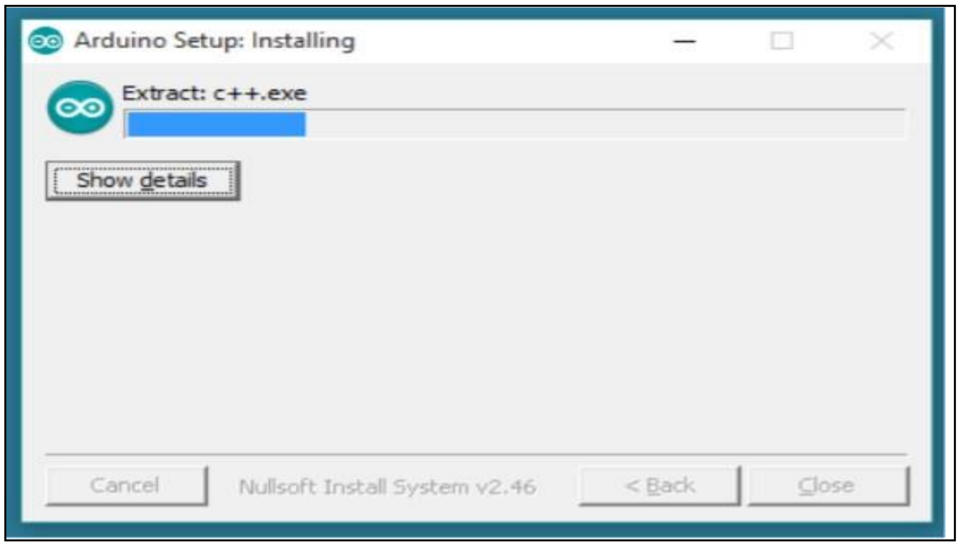
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino\

Space required: 392.7MB
Space available: 24.6GB

Step 3: Extraction of Files

➢ The process will extract and install all the required files to execute properlythe Arduino Software (IDE).


Arduino Setup: Installing

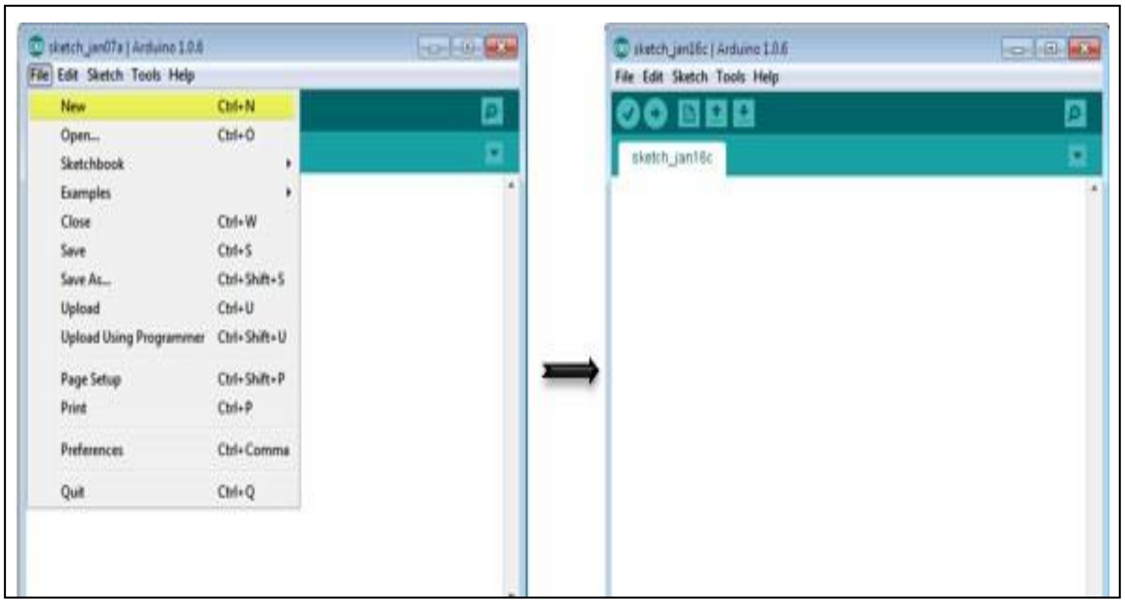
Extract: c++.exe

Step 4: Connecting the board

- The USB connection with the PC is necessary to program the board and not just to power it up. The Uno and Mega automatically draw power fromeither the USB or an external power supply. Connect the board to the computer using the USB cable. The green power LED (labelled PWR) should go on.

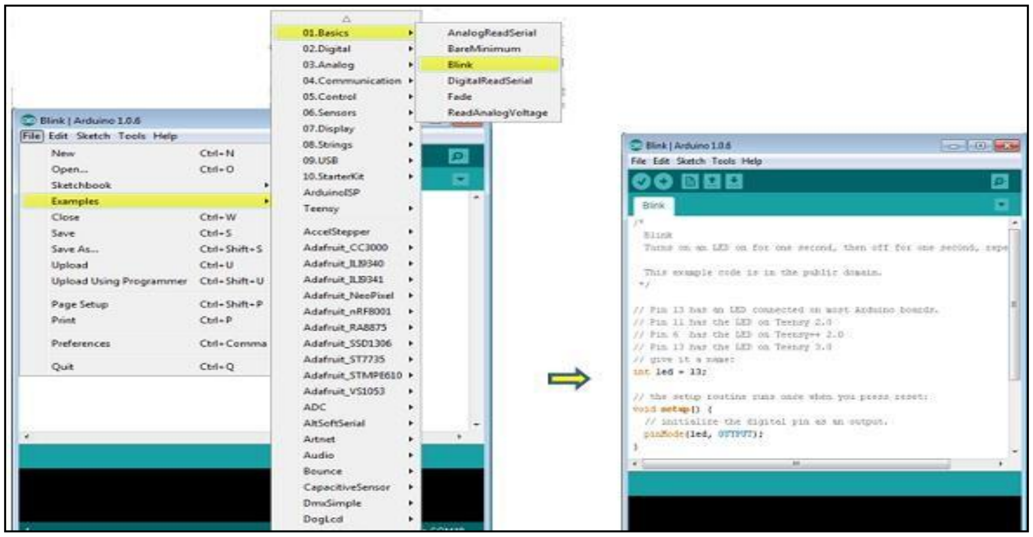
Step 5: Working on the new project

- Open the Arduino IDE software on your computer. Coding in the Arduinolanguage will control your circuit.
- Open a new sketch File by clicking on New.



Step 6: Working on an existing project

 To open an existing project example, select File → Example → Basics →Blink.

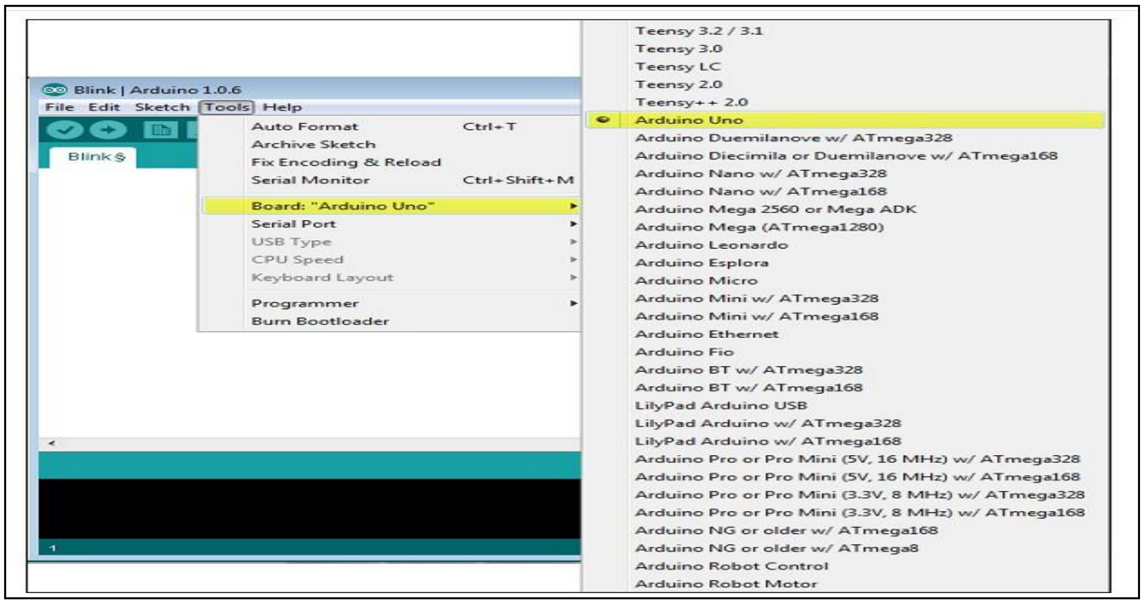


```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 has the LED on Teensy 2.0
// Pin 6  has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

Step 7: **Select your Arduino board.**

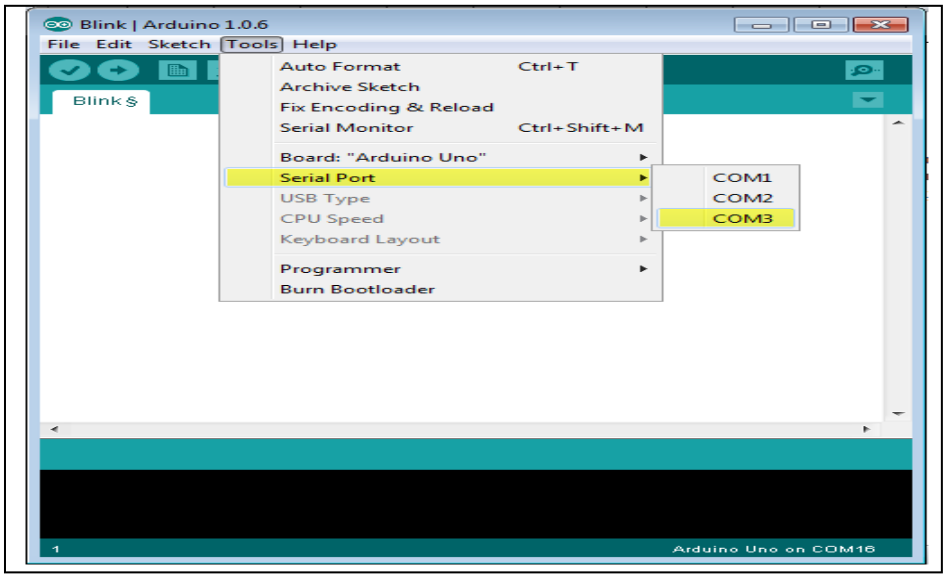
 To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

 Go to Tools → Board and select your board.


Blink | Arduino 1.0.6
File Edit Sketch Tools Help
Blink §
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Arduino Uno"
Serial Port
USB Type
CPU Speed
Keyboard Layout
Programmer
Burn Bootloader
Teensy 3.2 / 3.1
Teensy 3.0
Teensy LC
Teensy 2.0
Teensy++ 2.0
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Esplora
Arduino Micro
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino USB
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Arduino Robot Control
Arduino Robot Motor

Step 8: Select your serial port

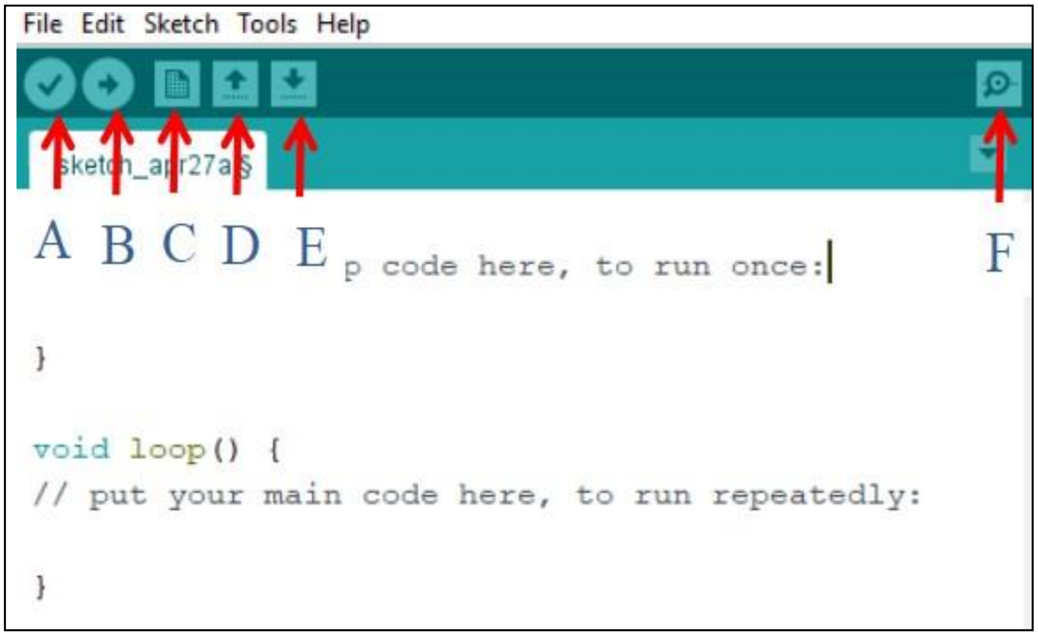
- Select the serial device of the Arduino board.
- Go to Tools → Serial Port menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports).
- To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the boardand select that serial port.



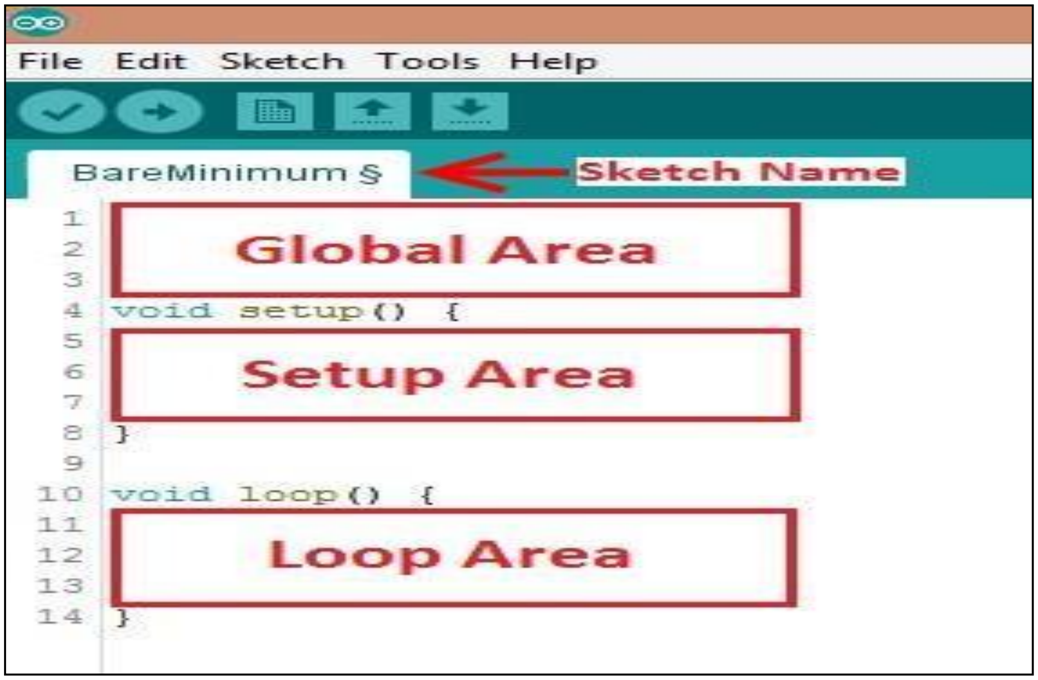
Step 9: Upload the program to your board.

- Click the "Upload" button in the environment.
- Wait a few seconds; you will see the RX and TX LEDs on the board, flashing.
- If the upload is successful, the message "Done uploading" will appear in thestatus bar.

A	Verify
B	Upload
C	New
D	Open
E	Save
F	Serial Motor



```
p code here, to run once:

}

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

}
```



```
void setup() {

}

void loop() {

}
```

{

BASICS OF C LANGUAGE USING ARDUINO IDE

STRUCTURE

The basic structure of the Arduino programming language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of

statements.

```
void setup()

{

 statements;

}

void loop()

{

 statements;

}
```

Where setup() is the preparation, loop() is the execution. Both functions are required for the program to work.

The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program, is run only once, and is used to set pinMode or initialize serial communication.

The loop function follows next and includes the code to be executed continuously – reading inputs, triggering outputs, etc. This function is the core of all Arduino programs and does the bulk of the work.

setup()

The setup() function is called once when your program starts. Use it to initialize pin modes, or begin serial. It must be included in a program even if there are no statements to run.

```
void setup()

{

 pinMode(pin, OUTPUT); // sets the 'pin' as output
}
```

loop()

After calling the setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing the program to change, respond, and control the Arduino board.

```
void loop()

{

 digitalWrite(pin, HIGH); // turns 'pin' on

 delay(1000); // pauses for one second

 digitalWrite(pin, LOW); // turns 'pin' off

 delay(1000); // pauses for one second

}
```

FUNCTIONS

A function is a block of code that has a name and a block of statements that are executed when the function is called. The functions void setup() and void loop() have already been discussed and other built-in functions will be discussed later. Custom functions can be written to perform repetitive tasks and reduce clutter in a program. Functions are declared by first declaring the function type. This is the type of value to be returned by the function such as 'int' for an integer type function. If no value is to be returned the function type would be void. After type, declare the name given to the function and in parenthesis any parameters being passed to the function.

```
type functionName(parameters)

{

 statements;

}
```

The following integer type function delayVal() is used to set a delay value in a program by reading the value of a potentiometer. It first declares a local variable v, sets v to the value of the potentiometer which gives a number between 0-1023, then divides that value by 4 for a final value between 0-255, and finally returns that value back to the main program.

```
int delayVal()

{
 int v; // create temporary variable 'v'

 v = analogRead(pot); // read potentiometer value

 v /= 4; // converts 0-1023 to 0-255

 return v; // return final value

}
```

{} CURLY BRACES

Curly braces (also referred to as just "braces" or "curly brackets") define the beginning and end of function blocks and statement blocks such as the void loop() function and the for and if statements.

```
type function()

{

 statements;

}
```

An opening curly brace { must always be followed by a closing curly brace }. This is often referred to as the braces being balanced. Unbalanced braces can often lead to cryptic, impenetrable compiler errors that can sometimes be hard to track down in a large program.

The Arduino environment includes a convenient feature to check the balance of curly braces. Just select a brace, or even click the insertion point immediately following a brace, and its logical companion will be highlighted.

; SEMICOLON

A semicolon must be used to end a statement and separate elements of the program. A semicolon is also used to separate elements in a for loop.

```
int x = 13; // declares variable 'x' as the integer 13
```

Note: Forgetting to end a line in a semicolon will result in a compiler error. The error text may be obvious, and refer to a missing semicolon, or it may not. If an impenetrable or seemingly illogical compiler error comes up, one of the first things to check is a missing semicolon, near the line where the compiler complained.

/*… */ BLOCK COMMENTS

Block comments, or multi-line comments, are areas of text ignored by the program and are used for large text descriptions of code or comments that help others understand parts of the program. They begin with /* and end with */ and can span multiple lines.

/* this is an enclosed block comment

 don’t forget the closing comment -

 they have to be balanced!

*/

Because comments are ignored by the program and take no memory space they should be used generously and can also be used to “comment out” blocks of code for debugging purposes.

Note: While it is possible to enclose single line comments within a block comment, enclosing a second block comment is not allowed.

// line comments

Single line comments begin with // and end with the next line of code. Like block comments, they are ignored by the program and take no memory space.

// this is a single line comment

Single line comments are often used after a valid statement to provide more information about what the statement accomplishes or to provide a future reminder.

VARIABLES

A variable is a way of naming and storing a numerical value for later use by the program. As their namesake suggests, variables are numbers that can be continually changed as opposed to constants whose value never changes. A variable needs to be declared and optionally assigned to the value needing to be stored. The following code declares a variable called inputVariable and then assigns it the value obtained

on analog input pin 2:

int inputVariable = 0; // declares a variable and

 // assigns value of 0

inputVariable = analogRead(2); // set variable to value of

 // analog pin 2

‘inputVariable’ is the variable itself. The first line declares that it will contain an int, short for integer. The second line sets the variable to the value at analog pin 2. This makes the value of pin 2 accessible elsewhere in the code.

Once a variable has been assigned, or re-assigned, you can test its value to see if it meets certain conditions, or you can use its value directly. As an example to illustrate three useful operations with variables, the following code tests whether the inputVariable is less than 100, if true it assigns the value 100 to inputVariable, and then sets a delay based on inputVariable which is now a minimum of 100:

```
if (inputVariable < 100) // tests variable if less than 100

{

 inputVariable = 100; // if true assigns value of 100

}

delay(inputVariable); // uses variable as delay
```

Note: Variables should be given descriptive names, to make the code more readable. Variable names like tiltSensor or pushButton help the programmer and anyone else reading the code to understand what the variable represents. Variable names like var or value, on the other hand, do little to make the code readable and are only used here as examples. A variable can be named any word that is not already one of the keywords in the Arduino language.

VARIABLE DECLARATION

All variables have to be declared before they can be used. Declaring a variable means defining its value type, as in int, long, float, etc., setting a specified name, and optionally assigning an initial value. This only needs to be done once in a program but the value can be changed at any time using arithmetic and various assignments.

The following example declares that inputVariable is an int, or integer type, and that its initial value equals zero. This is called a simple assignment.

```
int inputVariable = 0;
```

A variable can be declared in a number of locations throughout the program and where this definition takes place determines what parts of the program can use the variable.

VARIABLE SCOPE

A variable can be declared at the beginning of the program before void setup(), locally inside of functions, and sometimes within a statement block such as for loops.

Where the variable is declared determines the variable scope, or the ability of certain parts of a program to make use of the variable.

A global variable is one that can be seen and used by every function and statement in a program. This variable is declared at the beginning of the program, before the setup() function.

A local variable is one that is defined inside a function or as part of a for loop. It is only visible and can only be used inside the function in which it was declared. It is therefore possible to have two or more variables of the same name in different parts of the same program that contain different values.

Ensuring that only one function has access to its variables simplifies the program and reduces the potential for programming errors.

The following example shows how to declare a few different types of variables and demonstrates each variable’s visibility:

```
int value; // 'value' is visible

 // to any function

void setup()

{

 // no setup needed

}

void loop()

{

 for (int i=0; i<20;) // 'i' is only visible

 { // inside the for-loop

 i++;

 }

 float f; // 'f' is only visible

} // inside loop
```

BYTE

Byte stores an 8-bit numerical value without decimal points. They have a range of 0- 255.

```
byte someVariable = 180; // declares 'someVariable'

 // as a byte type
```

int

Integers are the primary datatype for storage of numbers without decimal points and store a 16-bit value with a range of 32,767 to -32,768.

```
int someVariable = 1500; // declares 'someVariable'
```

// as an integer type

Note: Integer variables will roll over if forced past their maximum or minimum values by an assignment or comparison. For example, if x = 32767 and a subsequent

statement adds 1 to x, x = x + 1 or x++, x will then rollover and equal -32,768.

LONG

Extended size datatype for long integers, without decimal points, stored in a 32-bit value with a range of 2,147,483,647 to -2,147,483,648.

long someVariable = 90000; // declares 'someVariable'

// as a long type

FLOAT

A datatype for floating-point numbers, or numbers that have a decimal point. Floatingpoint numbers have greater resolution than integers and are stored as a 32-bit value with a range of 3.4028235E+38 to -3.4028235E+38.

float someVariable = 3.14; // declares 'someVariable'

// as a floating-point type

Note: Floating-point numbers are not exact, and may yield strange results when compared. Floating point math is also much slower than integer math in performing calculations, so should be avoided if possible.

ARRAYS

An array is a collection of values that are accessed with an index number. Any value in the array may be called upon by calling the name of the array and the index number of the value. Arrays are zero indexed, with the first value in the array beginning at index number 0. An array needs to be declared and optionally assigned values before they can be used.

int myArray[] = {value0, value1, value2...}

Likewise it is possible to declare an array by declaring the array type and size and later assign values to an index position:

int myArray[5]; // declares integer array w/ 6 positions

myArray[3] = 10; // assigns the 4th index the value 10

To retrieve a value from an array, assign a variable to the array and index position:

```
x = myArray[3]; // x now equals 10
```

Arrays are often used in for loops, where the increment counter is also used as the index position for each array value. The following example uses an array to flicker an LED. Using a for loop, the counter begins at 0, writes the value contained at index position 0 in the array flicker[], in this case 180, to the PWM pin 10, pauses for 200ms, then moves to the next index position.

```
int ledPin = 10; // LED on pin 10

byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};

 // above array of 8

void setup() // different values

{

 pinMode(ledPin, OUTPUT); // sets OUTPUT pin

}

void loop()

{

 for(int i=0; i<7; i++) // loop equals number

 { // of values in array

 analogWrite(ledPin, flicker[i]); // write index value

 delay(200); // pause 200ms

 }

}
```

ARITHMETIC

Arithmetic operators include addition, subtraction, multiplication, and division. They return the sum, difference, product, or quotient (respectively) of two operands.

```
y = y + 3;

x = x - 7;
```

i = j * 6;

r = r / 5;

The operation is conducted using the data type of the operands, so, for example, 9 / 4 results in 2 instead of 2.25 since 9 and 4 are ints and are incapable of using decimal points. This also means that the operation can overflow if the result is larger than what can be stored in the data type. If the operands are of different types, the larger type is used for the calculation. For example, if one of the numbers (operands) are of the type float and the other of type integer, floating point math will be used for the calculation.

Choose variable sizes that are large enough to hold the largest results from your calculations. Know at what point your variable will rollover and also what happens in the other direction e.g. (0 - 1) OR (0 - - 32768). For math that requires fractions, use float variables, but be aware of their drawbacks: large size and slow computation speeds.

Note: Use the cast operator e.g. (int)myFloat to convert one variable type to another on the fly. For example, i = (int)3.6 will set i equal to 3.

COMPOUND ASSIGNMENTS

Compound assignments combine an arithmetic operation with a variable assignment. These are commonly found in for loops as described later. The most common compound assignments include:

x ++ // same as x = x + 1, or increments x by +1

x -- // same as x = x - 1, or decrements x by -1

x += y // same as x = x + y, or increments x by +y

x -= y // same as x = x - y, or decrements x by -y

x *= y // same as x = x * y, or multiplies x by y

x /= y // same as x = x / y, or divides x by y

Note: For example, x *= 3 would triple the old value of x and re-assign the resulting value to x.

COMPARISON OPERATORS

Comparisons of one variable or constant against another are often used in if statements to test if a specified condition is true. In the examples found on the following pages, ?? is used to indicate any of the following conditions:

x == y // x is equal to y

x != y // x is not equal to y

```
x < y // x is less than y

x > y // x is greater than y

x <= y // x is less than or equal to y

x >= y // x is greater than or equal to y
```

LOGICAL OPERATORS

Logical operators are usually a way to compare two expressions and return a TRUE or FALSE depending on the operator. There are three logical operators, AND, OR, and NOT, that are often used in if statements:

Logical AND:

```
if (x > 0 && x < 5) // true only if both

 // expressions are true
```

Logical OR:

```
if (x > 0 || y > 0) // true if either

 // expression is true
```

Logical NOT:

```
if (!x > 0) // true only if

 // expression is false
```

CONSTANTS

The Arduino language has a few predefined values, which are called constants. They are used to make the programs easier to read. Constants are classified in groups. true/false These are Boolean constants that define logic levels. FALSE is easily defined as 0 (zero) while TRUE is often defined as 1, but can also be anything else except zero. So in a Boolean sense, -1, 2, and -200 are all also defined as TRUE.

```
if (b == TRUE);

{

 doSomething;

}
```

HIGH/LOW

These constants define pin levels as HIGH or LOW and are used when reading or writing to digital pins. HIGH is defined as logic level 1, ON, or 5 volts while LOW is logic level 0, OFF, or 0 volts.

```
digitalWrite(13, HIGH);
```

input/output

Constants used with the pinMode() function to define the mode of a digital pin as either INPUT or OUTPUT.

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

if

if statements test whether a certain condition has been reached, such as an analog value being above a certain number, and executes any statements inside the brackets if the statement is true. If false the program skips over the statement. The format for an if test is:

```
if (someVariable ?? value)

{

 doSomething;

}
```

The above example compares someVariable to another value, which can be either a variable or constant. If the comparison, or condition in parentheses is true, the statements inside the brackets are run. If not, the program skips over them and continues on after the brackets.

Note: Beware of accidentally using ‘=’, as in if(x=10), while technically valid, defines the variable x to the value of 10 and is as a result always true. Instead use ‘==’, as in if(x==10), which only tests whether x happens to equal the value 10 or not. Think of ‘=’ as “equals” opposed to ‘==’ being “is equal to”.

IF… ELSE

if… else allows for ‘either-or’ decisions to be made. For example, if you wanted to test a digital input, and do one thing if the input went HIGH or instead do another thing if the input was LOW, you would write that this way:

```
if (inputPin == HIGH)

{

 doThingA;
}
else
{
 doThingB;
}
```

else can also precede another if test, so that multiple, mutually exclusive tests can be run at the same time. It is even possible to have an unlimited number of these else branches. Remember though, only one set of statements will be run depending on the condition tests:

```
if (inputPin < 500)
{
 doThingA;
}
else if (inputPin >= 1000)
{
 doThingB;
}
else
{
 doThingC;
}
```

Note: An if statement simply tests whether the condition inside the parenthesis is true or false. This statement can be any valid C statement as in the first example, if (inputPin == HIGH). In this example, the if statement only checks to see if indeed the specified input is at logic level high, or +5v.

for

The for statement is used to repeat a block of statements enclosed in curly braces a specified number of times. An increment counter is often used to increment and terminate the loop. There are three parts, separated by semicolons (;), to the for loop header:

```
for (initialization; condition; expression)

{

 doSomething;

}
```

The initialization of a local variable, or increment counter, happens first and only once. Each time through the loop, the following condition is tested. If the condition remains true, the following statements and expression are executed and the condition is tested again. When the condition becomes false, the loop ends.

The following example starts the integer i at 0, tests to see if i is still less than 20 and if true, increments i by 1 and executes the enclosed statements:

```
for (int i=0; i<20; i++) // declares i, tests if less

{ // than 20, increments i by 1

 digitalWrite(13, HIGH); // turns pin 13 on

 delay(250); // pauses for 1/4 second

 digitalWrite(13, LOW); // turns pin 13 off

 delay(250); // pauses for 1/4 second

}
```

Note: The C for loop is much more flexible than for loops found in some other computer languages, including BASIC. Any or all of the three header elements may be omitted, although the semicolons are required. Also the statements for initialization, condition, and expression can be any valid C statements with unrelated variables. These types of unusual for statements may provide solutions to some rare programming problems.

WHILE

while loops will loop continuously, and infinitely, until the expression inside the parenthesis becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

```
while (someVariable ?? value)

{

 doSomething;
}
```

The following example tests whether ‘someVariable’ is less than 200 and if true executes the statements inside the brackets and will continue looping until ‘someVariable’ is no longer less than 200.

```
while (someVariable < 200) // tests if less than 200

{

 doSomething; // executes enclosed statements

 someVariable++; // increments variable by 1

}
```

DO… WHILE

The do loop is a bottom driven loop that works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

```
do

{

 doSomething;

} while (someVariable ?? value);
```

The following example assigns readSensors() to the variable ‘x’, pauses for 50 milliseconds, then loops indefinitely until ‘x’ is no longer less than 100:

```
do

{

 x = readSensors(); // assigns the value of

 // readSensors() to x

 delay (50); // pauses 50 milliseconds

} while (x < 100); // loops if x is less than 100
```

Experiment 1:

Interfacing Light Emitting Diode(LED)- Blinking LED

Interfacing a Light Emitting Diode (LED) and making it blink is one of the simplest projects you can do with an Arduino. Below is a basic procedure for interfacing an LED and making it blink using an Arduino.

Components Required:

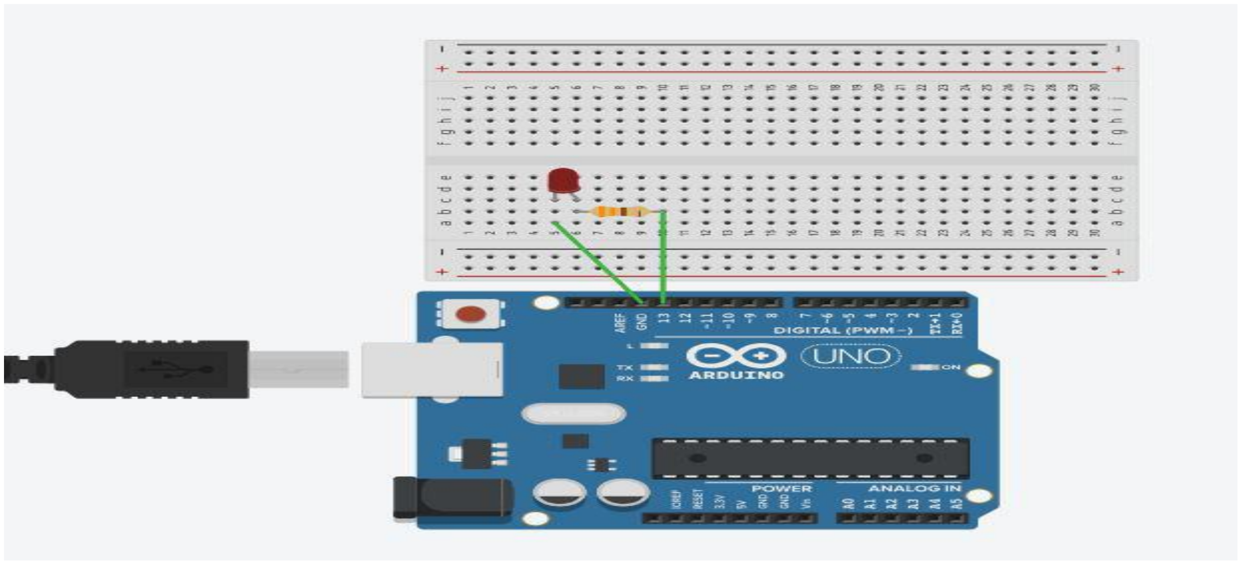
1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. LED (any color)
4. Resistor (220-330 ohms)

Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Copy and paste the provided Arduino code into the IDE.
5. Select the correct board and port from the Tools menu in the Arduino IDE.
6. Click the "Upload" button to upload the code to the Arduino.
7. Observe the LED on Pin 13 blinking on and off at a 1-second interval.

Circuit Connections:

- Connect the positive (longer leg) of the LED to a digital pin on the Arduino (Pin 8).
- Connect the negative (shorter leg) of the LED to a current-limiting resistor (220-330 ohms).
- Connect the other end of the resistor to the ground (GND) on the Arduino.



Arduino Code:

```
const int ledPin = 8; // Pin connected to the LED
void setup() {
  pinMode(ledPin, OUTPUT); // Set the LED pin as an output
}
void loop() {
  digitalWrite(ledPin, HIGH); // Turn on the LED
  delay(1000); // Wait for 1 second
  digitalWrite(ledPin, LOW);  // Turn off the LED
  delay(1000); // Wait for 1 second
}
```

Experiment 2:

Interfacing Button and LED – LED blinking when button is pressed

Components Required:

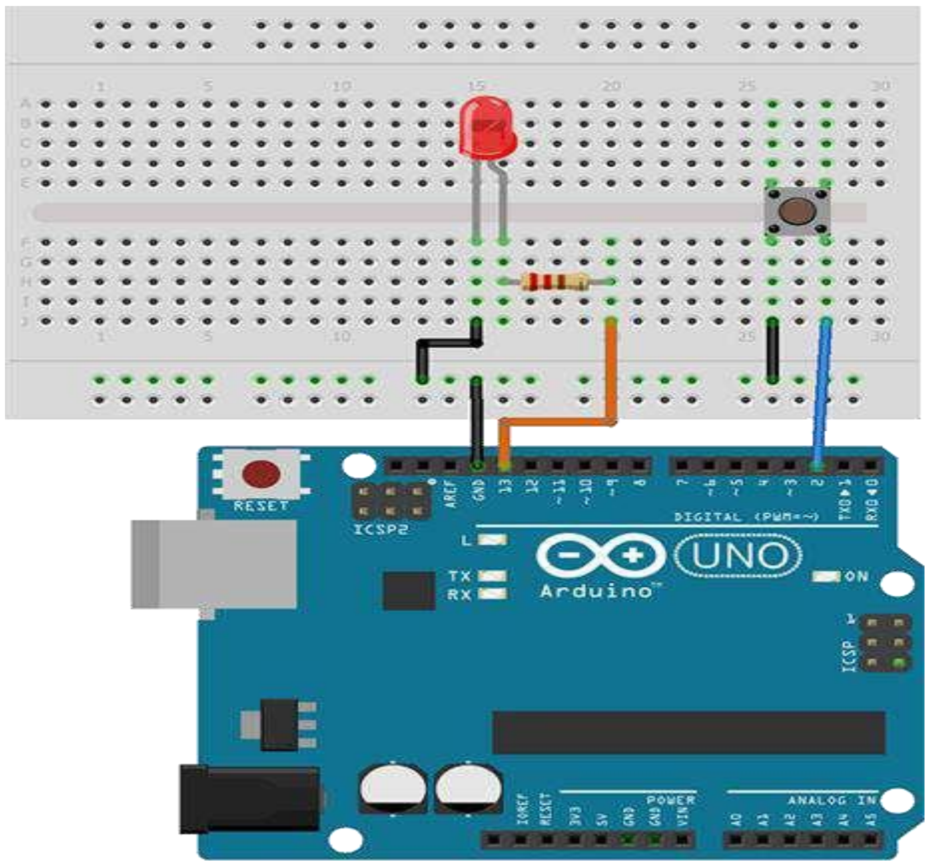
1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. LED (any color)
4. Resistor (220-330 ohms)
5. Push-button switch

Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Copy and paste the provided Arduino code into the IDE.
5. Select the correct board and port from the Tools menu in the Arduino IDE.
6. Click the "Upload" button to upload the code to the Arduino.
7. Once the upload is complete, press the button on the breadboard and observe the LED blinking in response to the button press.

Circuit Connections:

- Connect the positive (longer leg) of the LED to a digital pin on the Arduino (Pin 13).
- Connect the negative (shorter leg) of the LED to a current-limiting resistor (220-330 ohms).
- Connect the other end of the resistor to the ground (GND) on the Arduino.
- Connect one side of the push-button switch to another digital pin on the Arduino (Pin 2).
- Connect the other side of the push-button switch to the ground (GND) on the Arduino.
- Optionally, you can add a pull-up resistor (10k ohms) between the pin connected to the button and 5V on the Arduino.



Arduino Code:

```
const int buttonPin = 2; // Pin connected to the button
const int ledPin = 13;   // Pin connected to the LED

int buttonState = 0;     // Variable to store the button state

void setup() {
  pinMode(ledPin, OUTPUT);     // Set the LED pin as an output
  pinMode(buttonPin, INPUT);   // Set the button pin as an input
}

void loop() {
  buttonState = digitalRead(buttonPin); // Read the state of the button (HIGH or LOW)

  // Check if the button is pressed (buttonState is LOW)
  if (buttonState == LOW) {
    digitalWrite(ledPin, HIGH);  // Turn on the LED
    delay(500);                  // Wait for 500 milliseconds (0.5 seconds)
    digitalWrite(ledPin, LOW);   // Turn off the LED
    delay(500);                  // Wait for another 500 milliseconds
  }
}
```

Experiment 3:

Interfacing Light Dependent Resistor (LDR) and LED, displaying automatic night lamp

Creating an automatic night lamp using a Light Dependent Resistor (LDR) and LED is a common project to demonstrate light sensing. The LED turns on when it's dark (low light conditions) and turns off when it's bright. Here's a simple lab procedure for interfacing an LDR and LED to create an automatic night lamp.

Component Required:

1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. Light Dependent Resistor (LDR)
4. Resistor (10k ohms)
5. LED (any color)
6. Optional: Potentiometer (10k ohms) for sensitivity adjustment

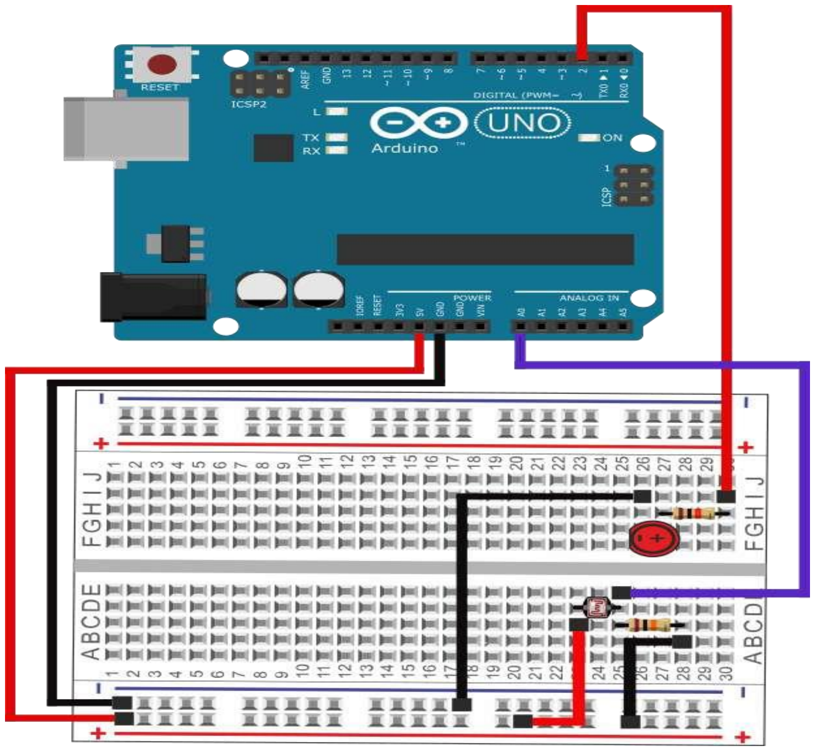
Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Copy and paste the provided Arduino code into the IDE.
5. Adjust the threshold value in the code based on your LDR sensitivity (you may need to experiment with different values).
6. Select the correct board and port from the Tools menu in the Arduino IDE.
7. Click the "Upload" button to upload the code to the Arduino.
8. Open the Serial Monitor to observe the LDR values and adjust the threshold accordingly.
9. Test the automatic night lamp by covering the LDR to simulate darkness.

Circuit Connections:

- Connect one leg of the LDR to the 5V on the Arduino.
- Connect the other leg of the LDR to the A0 (analog input) pin on the Arduino.
- Connect the junction of the LDR and the resistor to the ground (GND) on the Arduino.
- Connect one leg of the resistor (10k ohms) to the A0 pin on the Arduino.
- Connect the other leg of the resistor to the ground (GND) on the Arduino.
- Connect the positive (longer leg) of the LED to a digital pin on the Arduino
- Connect the negative (shorter leg) of the LED to a current-limiting resistor (220-330 ohms).

- Connect the other end of the resistor to the ground (GND) on the Arduino.



Arduino Code:

```
const int ldrPin = A0;   // Pin connected to the LDR
const int ledPin = 13;   // Pin connected to the LED
const int threshold = 500; // Adjust this value based on your LDR sensitivity

void setup() {
  pinMode(ledPin, OUTPUT); // Set the LED pin as an output
  Serial.begin(9600);      // Initialize serial communication for debugging
}

void loop() {
  int ldrValue = analogRead(ldrPin); // Read the LDR value (0-1023)
  Serial.println(ldrValue);          // Print LDR value to Serial Monitor for debugging

  // Check if it's dark (LDR value below threshold)
  if (ldrValue < threshold) {
    digitalWrite(ledPin, HIGH);  // Turn on the LED
  } else {
    digitalWrite(ledPin, LOW);   // Turn off the LED
  }
  delay(1000); // Delay for stability, adjust as needed
}
```

Experiment 4:

Interfacing Temperature Sensor(LM35) and/or humidity sensor (e.g.DHT11).

Interfacing a temperature sensor (such as LM35) and/or a humidity sensor (such as DHT11) with an Arduino is a common project in the realm of environmental sensing. Below, I'll provide a simple procedure for connecting both the LM35 temperature sensor and the DHT11 humidity sensor to an Arduino.

Component Required:

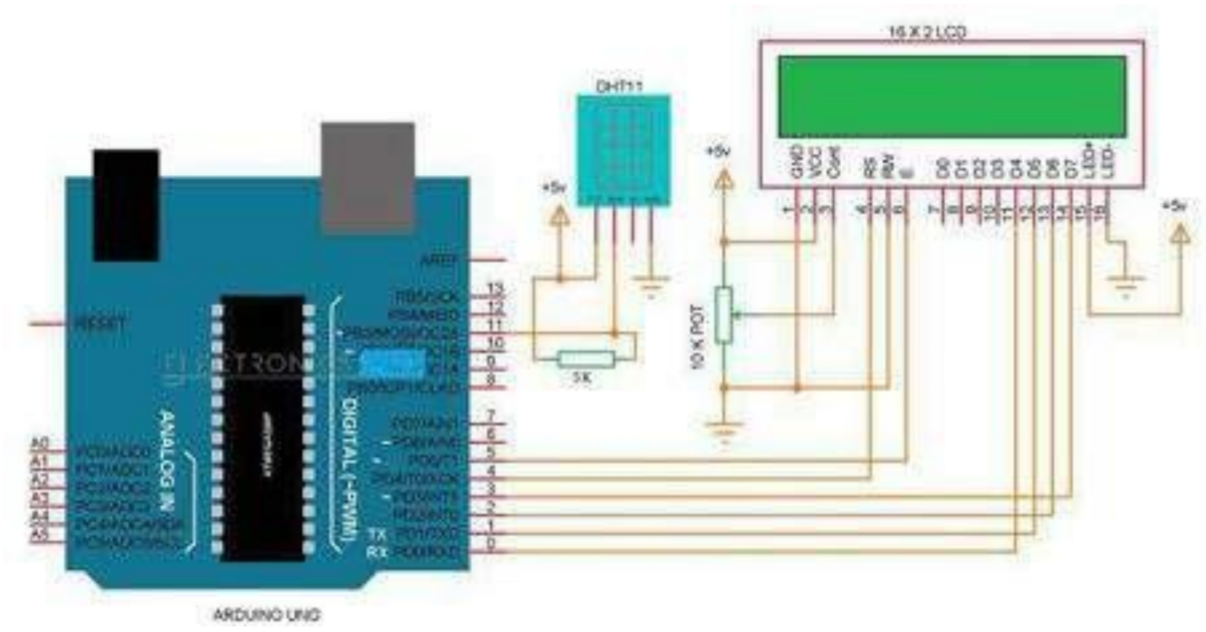
1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. LM35 temperature sensor
4. DHT11 humidity and temperature sensor (optional)
5. Resistors (if needed for DHT11)
6. Capacitor (if needed for DHT11)

Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Copy and paste the provided Arduino code into the IDE.
5. Select the correct board and port from the Tools menu in the Arduino IDE.
6. Click the "Upload" button to upload the code to the Arduino.
7. Open the Serial Monitor to observe the temperature and humidity readings.

Circuit Connections:

- For LM35 Temperature Sensor:
- Connect the LM35's VCC pin to 5V on the Arduino.
- Connect the LM35's GND pin to GND on the Arduino.
- Connect the LM35's OUT pin to an analog pin on the Arduino (A0).
- For DHT11 Humidity and Temperature Sensor:
- Connect the DHT11's VCC pin to 5V on the Arduino.
- Connect the DHT11's GND pin to GND on the Arduino.
- Connect the DHT11's DATA pin to a digital pin on the Arduino (D2).
- Optionally, add a pull-up resistor (5k ohms) between the VCC and DATA pins of the DHT11.



Arduino Code:

```
#include <DHT.h>

#define DHTPIN 2        // Digital pin connected to the DHT11
#define DHTTYPE DHT11   // DHT11 sensor model

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor

void setup() {
  Serial.begin(9600);   // Initialize serial communication for debugging
  dht.begin();          // Initialize DHT sensor
}

void loop() {
  // Read temperature from LM35
  int lm35Value = analogRead(A0);
  float temperatureLM35 = (lm35Value * 5.0 / 1023.0) * 100.0;

  // Read temperature and humidity from DHT11
  float temperatureDHT11 = dht.readTemperature();
  float humidityDHT11 = dht.readHumidity();

  // Print values to Serial Monitor
  Serial.print("LM35 Temperature: ");
  Serial.print(temperatureLM35);
  Serial.print(" °C\t");

  Serial.print("DHT11 Temperature: ");
  Serial.print(temperatureDHT11);
  Serial.print(" °C\t");

  Serial.print("DHT11 Humidity: ");
  Serial.print(humidityDHT11);
  Serial.println(" %");

  delay(2000); // Delay for stability, adjust as needed
}
```

Experiment 5:

Interfacing Liquid Crystal Display(LCD) – display data generated by sensor on LCD.

Interfacing a Liquid Crystal Display (LCD) with an Arduino to display data generated by a sensor is a common project in the field of electronics. Below is a simple procedure for connecting an LCD and displaying data from a sensor (e.g., LM35 temperature sensor).

Component Required:

1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. LM35 temperature sensor
4. LCD (16x2 or 20x4)
5. Potentiometer (10k ohms) for adjusting LCD contrast
6. Resistors (if needed for sensor)
7. Capacitors (if needed for sensor)

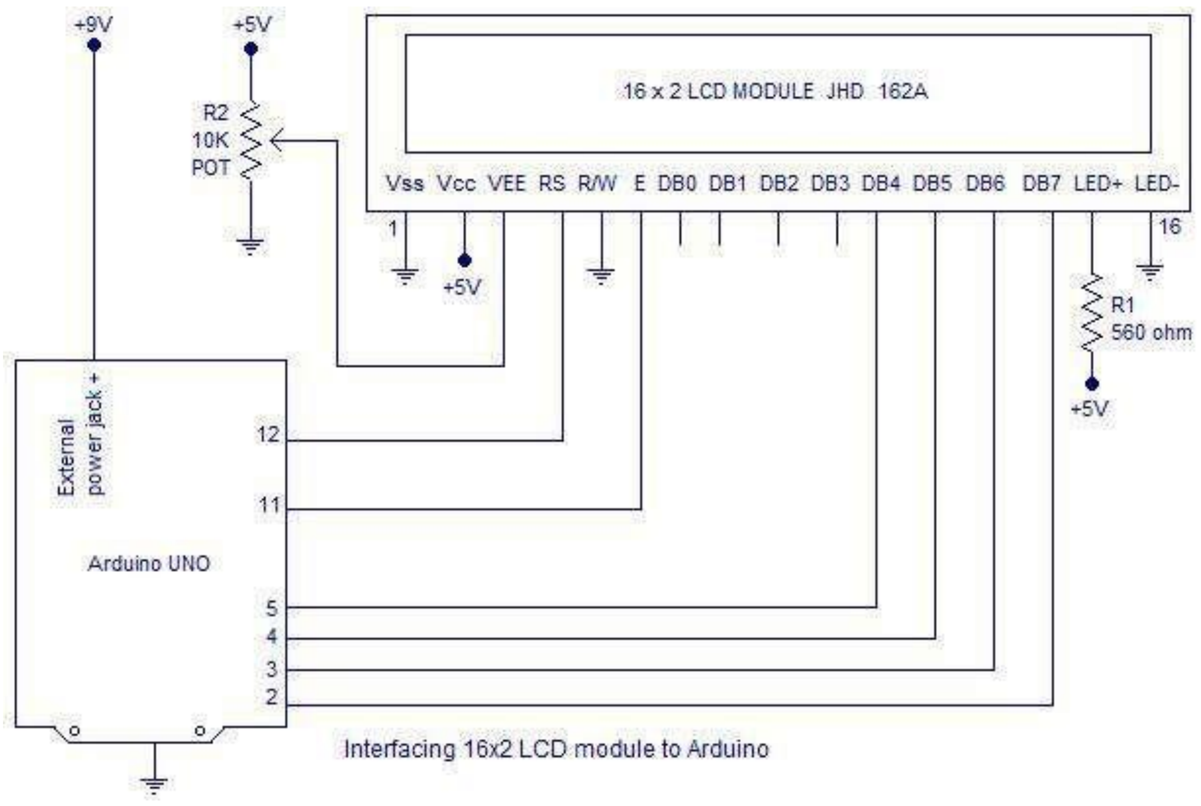
Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Install the "LiquidCrystal_I2C" library from the Library Manager in the Arduino IDE (Sketch -> Include Library -> Manage Libraries...).
5. Copy and paste the provided Arduino code into the IDE.
6. Select the correct board and port from the Tools menu in the Arduino IDE.
7. Click the "Upload" button to upload the code to the Arduino.
8. Adjust the potentiometer to set the LCD contrast.
9. Observe the temperature readings displayed on the LCD.

Circuit Connections:

- Connect the LM35 sensor to the Arduino as explained in the previous answer.
- Connect the LCD to the Arduino as follows:
- Connect the VCC pin of the LCD to 5V on the Arduino.
- Connect the GND pin of the LCD to GND on the Arduino.
- Connect the SDA pin of the LCD to a digital pin on the Arduino (A4).
- Connect the SCL pin of the LCD to another digital pin on the Arduino (A5).
- Connect the RW pin of the LCD to GND to set it in write mode.
- Connect the RS pin of the LCD to a digital pin on the Arduino (D7).
- Connect the EN pin of the LCD to a digital pin on the Arduino (D6).

- Connect the K pin of the LCD to GND.
- Connect the A pin of the LCD to 5V.
- Connect the wiper (center) pin of the potentiometer to the V0 pin on the LCD.


+9V
+5V
R2
10K
POT
16 x 2 LCD MODULE JHD 162A
Vss Vcc VEE RS R/W E DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7 LED+ LED-
1
16
+5V
R1
560 ohm
+5V
External power jack +
Arduino UNO
12
11
5
4
3
2
Interfacing 16x2 LCD module to Arduino

Arduino Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Initialize the LCD library with the I2C address
LiquidCrystal_I2C lcd(0x27, 16, 2);

const int lm35Pin = A0; // Analog pin connected to the LM35 sensor

void setup() {
  lcd.begin(16, 2);  // Initialize the LCD with 16 columns and 2 rows
  lcd.backlight();   // Turn on the backlight
  Serial.begin(9600); // Initialize serial communication for debugging
}

void loop() {
  // Read temperature from LM35
  int lm35Value = analogRead(lm35Pin);
  float temperature = (lm35Value * 5.0 / 1023.0) * 100.0;

  // Display temperature on the LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temperature:");
  lcd.setCursor(0, 1);
  lcd.print(temperature);
  lcd.print(" C");

  // Print temperature to Serial Monitor for debugging
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");

  delay(2000); // Delay for stability, adjust as needed
}
```

Experiment 6:

Interfacing Air Quality Sensor-pollution (e.g. MQ135) – display data on LCD switch on LED when data sensed is higher than specified value.

Interfacing an Air Quality Sensor (e.g., MQ135) with an Arduino, displaying data on an LCD, and activating an LED when the sensed pollution level exceeds a specified threshold is a valuable project for monitoring air quality. Below is a step-by-step procedure for setting up this project.

Component Required:

1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. MQ135 Air Quality Sensor
4. LCD (16x2 or 20x4)
5. Potentiometer (10k ohms) for adjusting LCD contrast
6. LED
7. Resistor (220-330 ohms) for the LED
8. Resistors (if needed for MQ135)
9. Capacitors (if needed for MQ135)

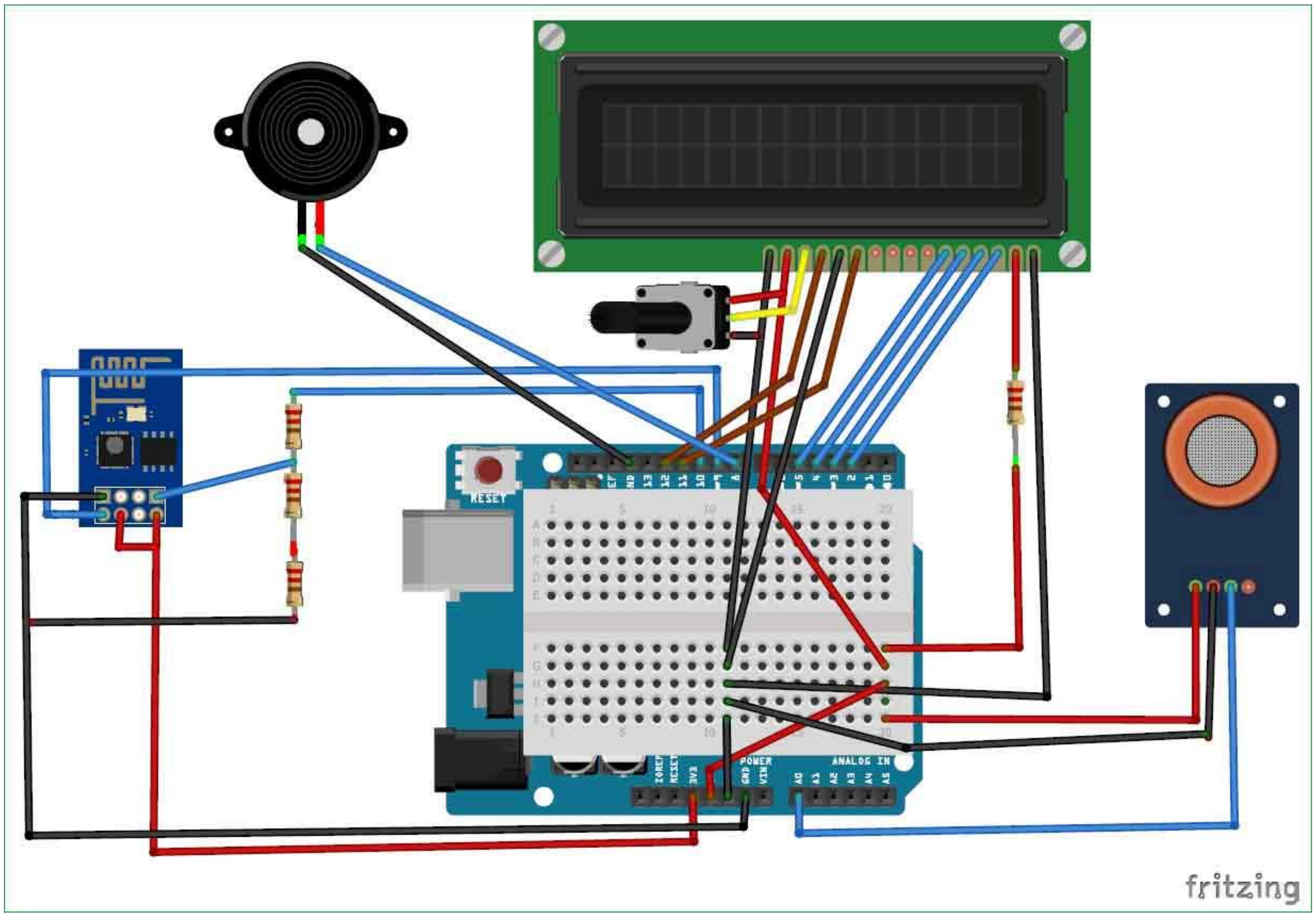
Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Install the "LiquidCrystal_I2C" library from the Library Manager in the Arduino IDE (Sketch -> Include Library -> Manage Libraries...).
5. Copy and paste the provided Arduino code into the IDE.
6. Select the correct board and port from the Tools menu in the Arduino IDE.
7. Click the "Upload" button to upload the code to the Arduino.
8. Adjust the potentiometer to set the LCD contrast.
9. Observe the air quality readings displayed on the LCD, and the LED should turn on when the pollution level exceeds the specified threshold.

Circuit Connections:

- Connect the MQ135 sensor to the Arduino:
- Connect the VCC pin of the MQ135 to 5V on the Arduino.
- Connect the GND pin of the MQ135 to GND on the Arduino.
- Connect the OUT pin of the MQ135 to an analog pin on the Arduino (A0).

- Connect the LCD to the Arduino as described in the previous answer.
- Connect the LED to the Arduino:
- Connect the anode (longer leg) of the LED to a digital pin on the Arduino (D7).
- Connect the cathode (shorter leg) of the LED to a current-limiting resistor (220-330 ohms).
- Connect the other end of the resistor to GND on the Arduino.



Arduino Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Initialize the LCD library with the I2C address
LiquidCrystal_I2C lcd(0x27, 16, 2);

const int mq135Pin = A0; // Analog pin connected to the MQ135 sensor
const int ledPin = 7;    // Digital pin connected to the LED
const int thresholdValue = 200; // Set your pollution threshold value
```

```
void setup() {
  lcd.begin(16, 2);    // Initialize the LCD with 16 columns and 2 rows
  lcd.backlight();     // Turn on the backlight
  pinMode(ledPin, OUTPUT); // Set the LED pin as an output
  Serial.begin(9600);  // Initialize serial communication for debugging
}

void loop() {
  // Read pollution level from MQ135
  int mq135Value = analogRead(mq135Pin);

  // Display pollution level on the LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Air Quality:");
  lcd.setCursor(0, 1);
  lcd.print(mq135Value);

  // Check if pollution level is higher than the threshold
  if (mq135Value > thresholdValue) {
    digitalWrite(ledPin, HIGH); // Turn on the LED
  } else {
    digitalWrite(ledPin, LOW);  // Turn off the LED
  }

  // Print pollution level to Serial Monitor for debugging
  Serial.print("Air Quality: ");
  Serial.println(mq135Value);

  delay(2000); // Delay for stability, adjust as needed
}
```

Experiment 7:

Interfacing Bluetooth module (e.g. HC05)- receiving data from mobile phone on Arduino and display on LCD.

Interfacing a Bluetooth module (such as HC-05) with an Arduino to receive data from a mobile phone and displaying it on an LCD is a popular project. Below is a step-by-step procedure for setting up this project.

Component Required:

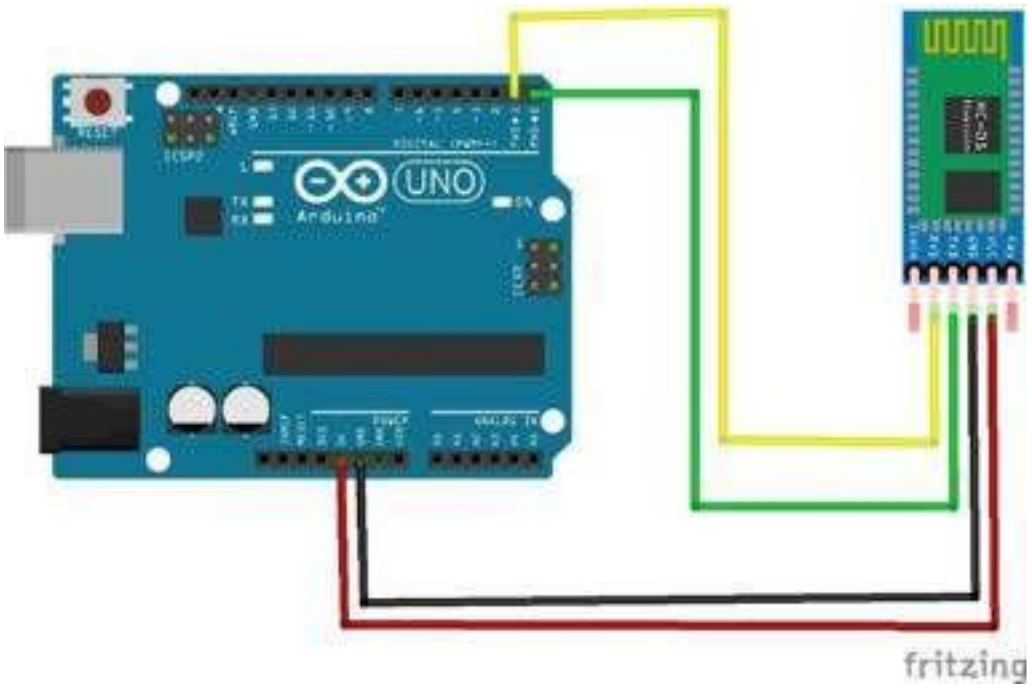
1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. Bluetooth module (HC-05)
4. LCD (16x2 or 20x4)
5. Potentiometer (10k ohms) for adjusting LCD contrast

Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Install the "Wire" and "LiquidCrystal_I2C" libraries from the Library Manager in the Arduino IDE (Sketch -> Include Library -> Manage Libraries...).
5. Copy and paste the provided Arduino code into the IDE.
6. Select the correct board and port from the Tools menu in the Arduino IDE.
7. Click the "Upload" button to upload the code to the Arduino.
8. Connect the Bluetooth module to your mobile phone using an appropriate app (e.g., Bluetooth Terminal or Serial Bluetooth Terminal).
9. Send data from the mobile app, and the received data should be displayed on the LCD connected to the Arduino.

Circuit Connections:

- Connect the Bluetooth module to the Arduino:
- Connect the VCC pin of the HC-05 to 5V on the Arduino.
- Connect the GND pin of the HC-05 to GND on the Arduino.
- Connect the TXD pin of the HC-05 to a digital pin on the Arduino (e.g., D2).
- Connect the RXD pin of the HC-05 to another digital pin on the Arduino (e.g., D3).
- Connect the EN pin of the HC-05 to 5V through a voltage divider (two resistors, e.g., 10k and 20k ohms).
- Connect the LCD to the Arduino



Arduino Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SoftwareSerial.h>

// Initialize the LCD library with the I2C address
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Initialize the SoftwareSerial library for Bluetooth communication
SoftwareSerial bluetoothSerial(2, 3); // RXD (HC-05 TXD) to D2, TXD (HC-05 RXD) to D3

void setup() {
  lcd.begin(16, 2);    // Initialize the LCD with 16 columns and 2 rows
  lcd.backlight();     // Turn on the backlight
  Serial.begin(9600);  // Initialize serial communication for debugging
  bluetoothSerial.begin(9600); // Initialize Bluetooth serial communication
}

void loop() {
  // Check if data is available from Bluetooth
  if (bluetoothSerial.available() > 0) {
    // Read the data from Bluetooth
    char data = bluetoothSerial.read();

    // Display the received data on the LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Received Data:");
    lcd.setCursor(0, 1);
    lcd.print(data);

    // Print received data to Serial Monitor for debugging
    Serial.print("Received Data: ");
    Serial.println(data);
  }
}
```

Experiment 8:

Interfacing Relay module to demonstrate Bluetooth based home automation application. (using Bluetooth and relay).

Creating a Bluetooth-based home automation application using an Arduino, a Bluetooth module (e.g., HC-05), and a relay module is a common and useful project. Below is a step-by-step procedure for setting up a basic home automation system that can control a relay through Bluetooth.

Component Required:

1. Arduino board (e.g., Arduino Uno)
2. Breadboard and jumper wires
3. Bluetooth module (HC-05)
4. Relay module
5. Devices/appliances to control (make sure they are compatible with the relay)

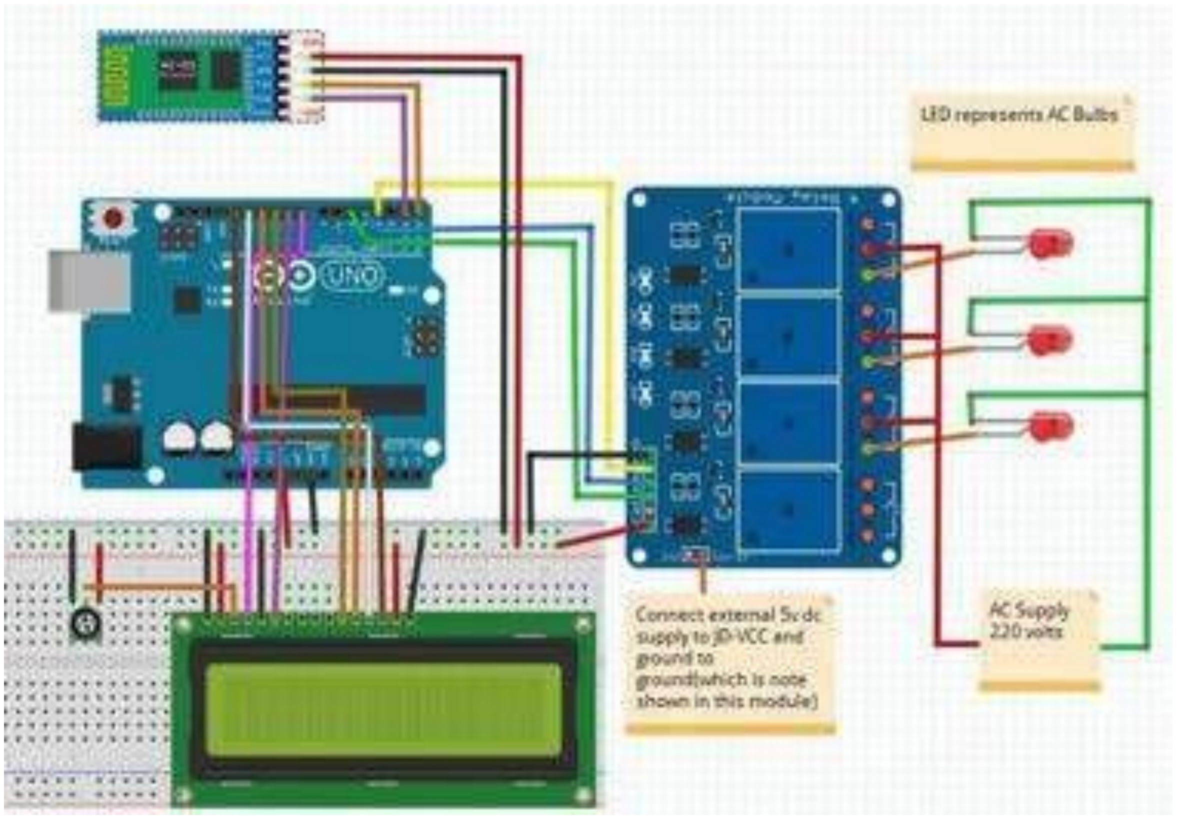
Procedure:

1. Connect the components on the breadboard according to the circuit connections mentioned above.
2. Connect the Arduino to your computer using a USB cable.
3. Open the Arduino IDE on your computer.
4. Copy and paste the provided Arduino code into the IDE.
5. Select the correct board and port from the Tools menu in the Arduino IDE.
6. Click the "Upload" button to upload the code to the Arduino.
7. Connect the Bluetooth module to your mobile phone using an appropriate app (e.g., Bluetooth Terminal or Serial Bluetooth Terminal).
8. Send '1' to turn on the relay or '0' to turn it off using the mobile app.

Circuit Connections:

- Connect the Bluetooth module to the Arduino:
- Connect the VCC pin of the HC-05 to 5V on the Arduino.
- Connect the GND pin of the HC-05 to GND on the Arduino.
- Connect the TXD pin of the HC-05 to a digital pin on the Arduino (e.g., D2).
- Connect the RXD pin of the HC-05 to another digital pin on the Arduino (e.g., D3).
- Connect the EN pin of the HC-05 to 5V through a voltage divider (two resistors, e.g., 10k and 20k ohms).
- Connect the relay module to the Arduino:
- Connect the VCC pin of the relay module to 5V on the Arduino.
- Connect the GND pin of the relay module to GND on the Arduino.

- Connect the IN or signal pin of the relay module to a digital pin on the Arduino (e.g., D7).
- Connect the devices/appliances to the relay module.


LED represents AC Bulbs
Connect external 5v dc supply to JD-VCC and ground to ground(which is note shown in this module)
AC Supply 220 volts
UNO

Arduino Code:

```
#include <SoftwareSerial.h>

SoftwareSerial bluetoothSerial(2, 3); // RXD (HC-05 TXD) to D2, TXD (HC-05 RXD) to D3

const int relayPin = 7; // Digital pin connected to the relay module

void setup() {
  pinMode(relayPin, OUTPUT); // Set the relay pin as an output
  Serial.begin(9600); // Initialize serial communication for debugging
  bluetoothSerial.begin(9600); // Initialize Bluetooth serial communication
}

void loop() {
  // Check if data is available from Bluetooth
  if (bluetoothSerial.available() > 0) {
    // Read the data from Bluetooth
    char command = bluetoothSerial.read();
    // Process the command received
    if (command == '1') {
      digitalWrite(relayPin, HIGH); // Turn on the relay
      Serial.println("Relay turned ON");
    } else if (command == '0') {
      digitalWrite(relayPin, LOW); // Turn off the relay
      Serial.println("Relay turned OFF");
    }
  }
}
```