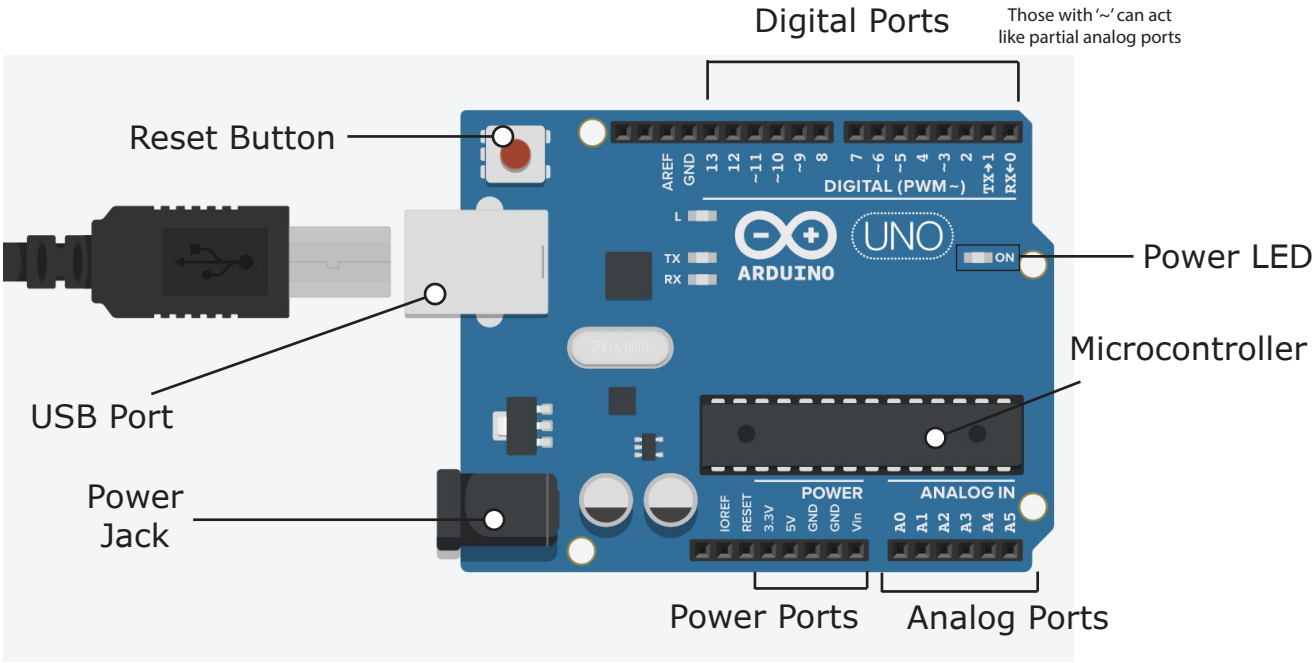
Arduino UNO

Lalith Kumar Reddy
ENIGMA, IIIT B

Arduino is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices. .... according to Wikipedia.

Right now, our main focus will be on the Arduino UNO.



These are some of the important parts of the board that you are supposed to know.

The USB port acts as a data transfer port as well as a power source. The board can also be powered from the power jack or the Vin port in the power ports line (at the end). The power LED glows when the board is powered.

The digital ports can produce only digital signals and analog ports can read only analog signals. But there are certain digital ports with the sign '~'. These ports can partially act as analog ports.

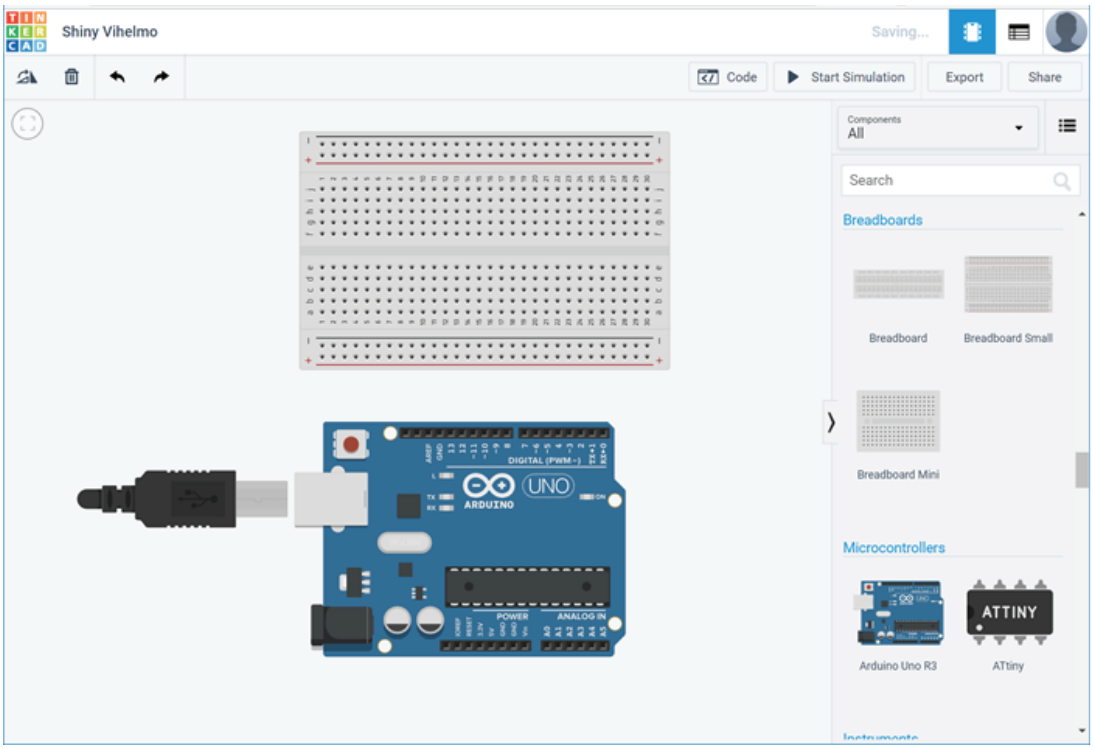
More on these soon.

Before we get into any programming, let me first introduce you to the environments where we will work with Arduino.

Firstly, lets visit Tinkercad.

Tinkercad is an online simulation platform for testing the circuits virtually. Before you run any doubtful programs on the real circuit, it is advised to test on Tinkercad first. It reduces the risk of busting the circuit and the pain to find the problem with the real one.

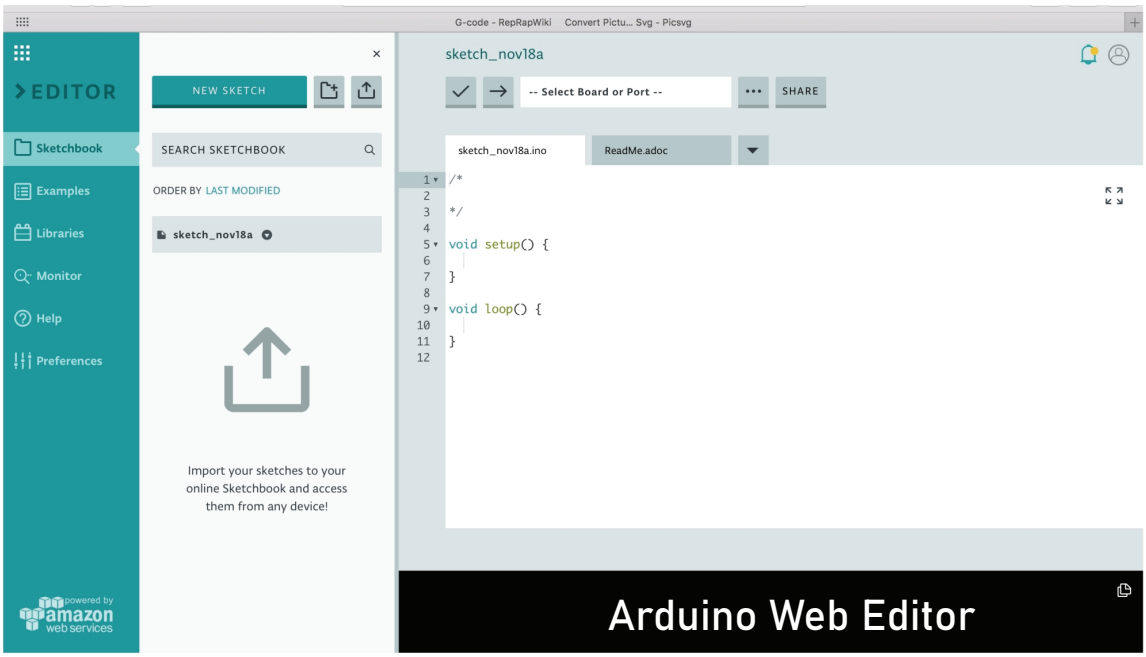
This is how the workspace looks like.



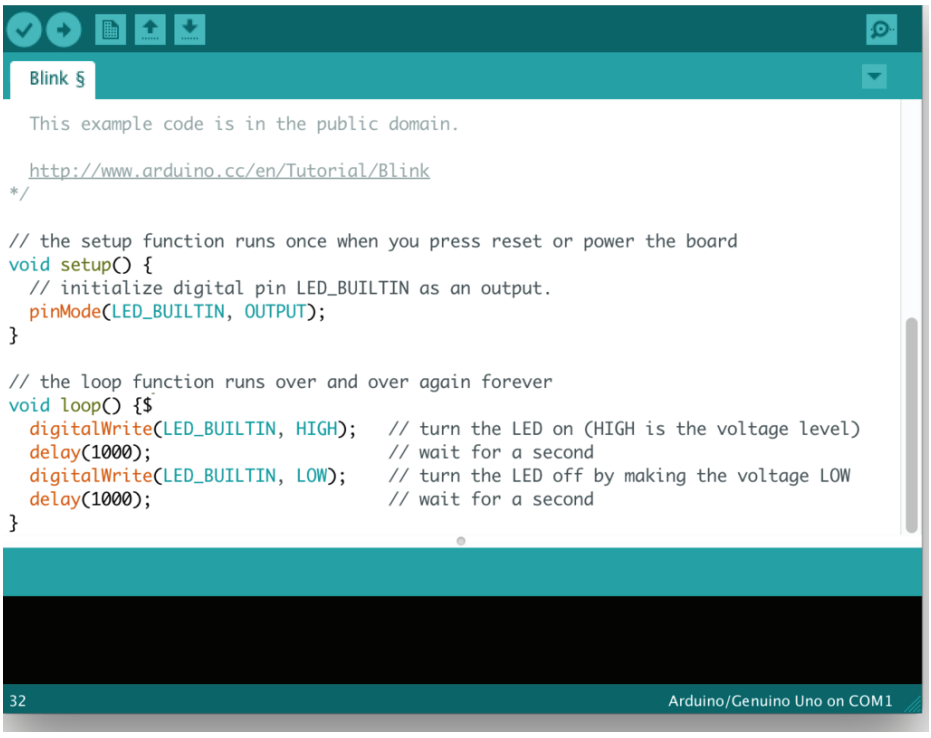
You shall be working with this for sometime before you start building actual circuits. The only way to understand any software is to start working with it. So go ahead and explore!

The next most important software we will use is Arduino IDE. You can use the online editor or you can download the application.

Either ways, the UI of both are pretty much the same.



Arduino Web Editor



Arduino IDE

We will be using this as our primary software to build all cicuits and robots. Most syntax maybe new to you but eveything else is exactly the same as C.

By default, whenever you open a new file in the editor, you will find something like this on your screen.

```
void setup()
{

}

void loop()
{

}
```

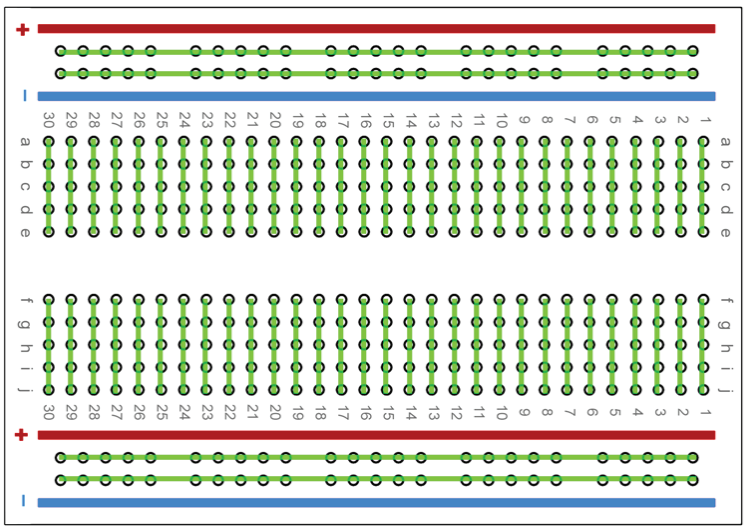
The setup is where you enable the pins you want use and loop is where you command the pins to read or write values.

First of all, your task is to light up an LED.
You can try this on Tinkercad.

Use the power pins to light an LED. Use a breadboard for convenience.

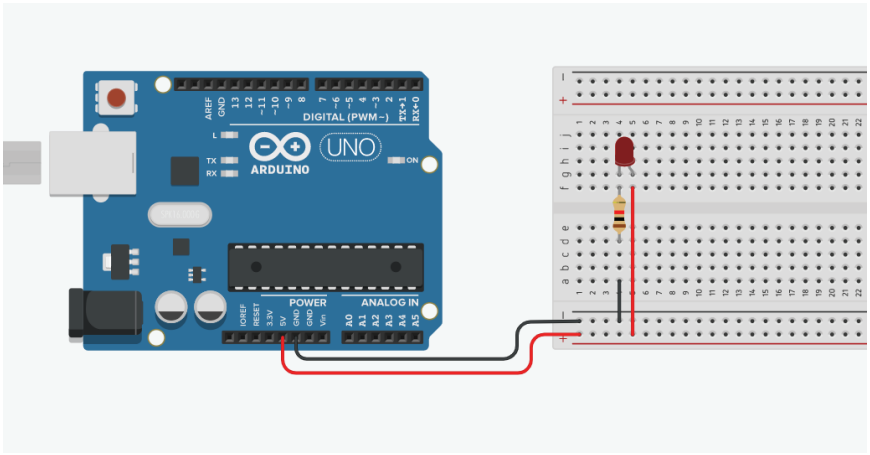
WARNING!: Always use a resistor in series to the LED! Or else the LED is as good as dead.

By the way, here is how a breadboard connections look.



Every green line you see is like one wire. All the holes in one green line are connected.

Here is what it should look like.



Now lets try to blink the LED. But to do so, we can use the digital pins. You will first have to setup which pin to drive it with. For that, in the setup function, we have to write this line.

```
pinMode(13, OUTPUT);   // pin_num, I/O type
```

And then to toggle it, set it high/low using this in the loop function.

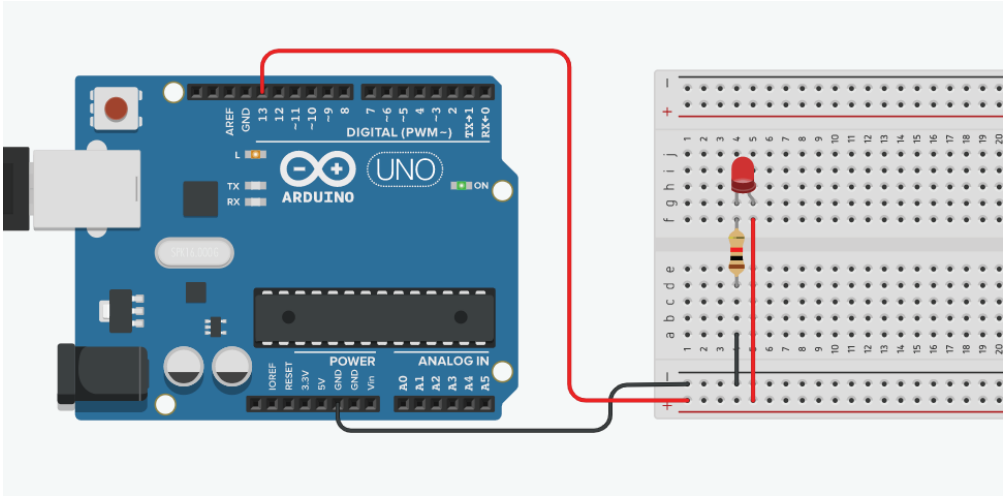
```
digitalWrite(13, HIGH);// pin_num, state
digitalWrite(13, LOW);
```

But this doesn't seem to have any visible effect. This is because there is no delay between the two states. For this, we just use

```
delay(500);// Units in microseconds (1s = 1000ms)
```

Write this after each of these two lines. Now you will see it blink. Mind you, setup is only to declare which pins to use. What to do with those pins is written in the loop function.

This is what the new connection should look like.



Here is what you code should be like.

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(13, LOW);
  delay(500); // Wait for 500 millisecond(s)
}
```

Now your task will be to make a travelling light dot. You should use LEDs in a line and make them blink one after the other.

For example, if there are 8 LEDs, first light up LED 1, and then turn it off and light up LED 2, and then turn that off and so on. When you reach LED 8, start it all over again.

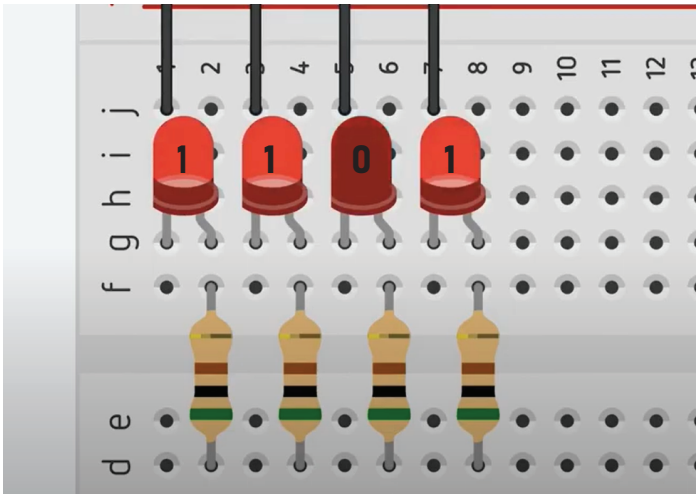
HINT: Use a while/for loop.

Pseudocode->

```
void setup()
{
  loop 1 to 8
    setup pin i for output;
}

void loop()
{
  loop 1 to 8{
    set i high
    wait
    set i low
  }
}
```

Your next task will be to make a binary counter using 4 LEDs. Each 1 should light up the respective LED and each 0 should be off.
You should start from 0 and go upto 15.
In this case, you can declare global variables to keep track of the step or limit.

Here is an example setup for your reference.



$1101_2 = 13_{10}$

Once you are done with that, try expanding it to 8 bits.

Now that you've done the simple blinking, is it possible to control the brightness of an LED? Can we make it light up slowly and then dim slowly?
But digitalWrite provides two states only, high or low. This is why we will have to use analogWrite.

analogWrite lets us set specific output values ranging from 0-255.
So the statement will look something like this.

```
analogWrite(<pin_num>, <value>);
```

The pins that can do this are the partial analog pins i.e. the digital pins with '~' (PWM). The value range, as mentioned, is limited from 0 to 255. If you try giving a value greater than 255, it remains high.

Now with that in mind, try to make the LED light up slowly and then dim again slowly. And do this without using any for/while loop. Use the given loop function to your advantage.

A simple way to do this will be to have a variable that defines rising or falling, one to keep track of the step (brightness) of the LED.

This is the code for you reference.

```
int state = 1; // Rising-1, Falling-0
int step = 0;

void setup()
{
  pinMode(3, OUTPUT);
}
void loop()
{
  if(state == 1){
    analogWrite(3, step);
    if(step < 255) step++;
    else   state = 0;
    delay(10);
  }
  else{
    analogWrite(3, step);
    if(step > 0)   step--;
    else   state = 1;
    delay(10);
  }
}
```

Execute this code to see it work clearly.

Now think of another way to implement this. This time with ONLY ONE if statement.

HINT: Toggle the state when you reach the end.

Your task on this will be to create a circuit such that there are 6 LEDs and the alternative LEDs should slowly light up and fade. And then the next set should light up and fade and continue.

For example, LEDs 1,3,5 should slowly light up and fade, all together. And then LEDs 2,4,6 shoud slowly light up and fade. Continue this over and over.

If possible, do so with (1,4), (2,5), (3,6) as well.

Lets see what else we can do with analogWrite. If analogWrite helps to set output values to control the brightness of the LED, then it should also be able to control the speed of the motor.

For the motor, it is prefered to use a motor driver. But it should still work well with just the arduino board.
For a motor, we require two PWM pins. The two PWM pins are connected to the motor. The direction in which the motor rotates depends on the connections you made.

Now set pin A at 255 and the pin B to 0. Without loss of generality, lets say it turns clockwise. If we reduce pin A to 127, the speed of the motor falls to half.
Now if we switch the values of A and B, the motor starts to rotate anti-clockwise. You will notice a similar effect if change the values.

The speed of the motor depends on the difference between the two pins. Larger the difference, faster it rotates.

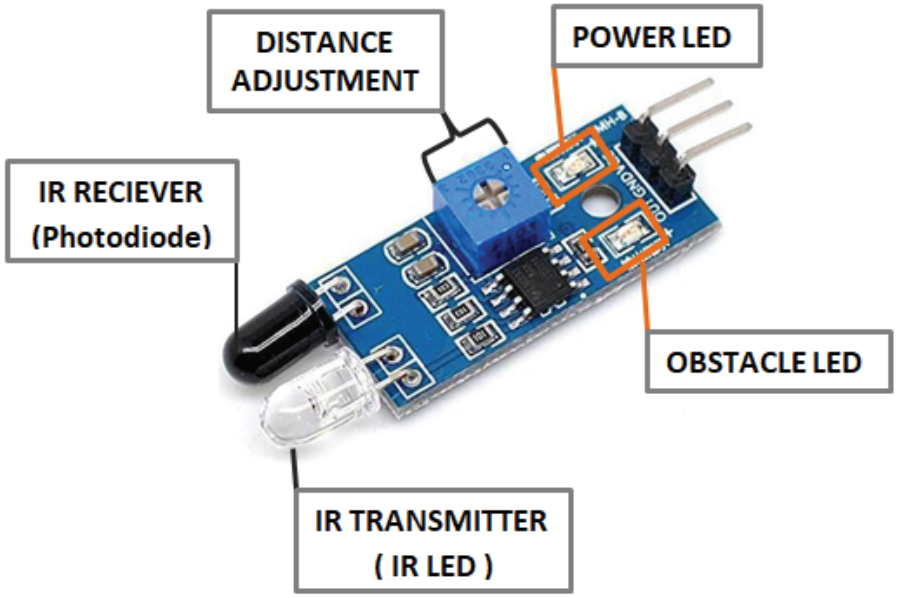
Here is the code segment for you to experiment with.

```
analogWrite(10, 255);
analogWrite(11, 0);
delay(2000);

analogWrite(10, 64);
analogWrite(11, 0);
delay(2000);

analogWrite(10, 0);
analogWrite(11, 255);
delay(2000);

analogWrite(10, 0);
analogWrite(11, 64);
delay(2000);

analogWrite(10, 0);
analogWrite(11, 0);
delay(2000);
```

Your task with the motors will be to build a simple robot that moves in a striaght line. Key to this is to set both the motors rotating in the right orientation and setting the speed equally.

Now lets check out the InfraRed Sensor. An IR sensor is used to detect the distance between an obstacle and the sensor (usually mounted on the robot).
It can also be used to detect difference in colour. In fact, it differentiates the black tape from the floor which we are gonna use for the line following robot. The black tape absorbs infrared radiation.

This is a diagram of an IR sensor and its parts.



The transmitter transmits Infrared waves and the receiver detects the waves that bounce back. If the receiver detects an obstacle, the obstacle LED glows.

There is a calibration screw which adjusts the sensitivity of the sensor. Usually its range is between half-a-feet to ~0.5cm.

We can read the output data using digitalRead.

digitalRead receives an output of 0(no detection) or 1(detection). We can use any digital pin to read the data. To check if the board detects any output from the sensor, make an LED glow.
In fact, there is an output LED corresponding to pin 13 on the board. So you can use it to check if the board receives the signals.

Connections->
VCC to 5V
GND to GND
OUT to input pin

Set the calibration screw for a reasonable distance. Do not turn it more than its limit. It is very easy to break it than to adjust it.

But to monitor the output from the source constantly, we need to use a variable to store the value. Also, the pin which you use should be set as input.

```
pinMode(10, INPUT);  // In setup
val = digitalRead(10);   // In loop
```

Hence, this is what the final code should be like.

```
int state = 0;

void setup()
{
  pinMode(13, OUTPUT);
  pinMode(10, INPUT);
}

void loop()
{
  state = digitalRead(10);
  digitalWrite(13, state);
  delay(50);
}
```

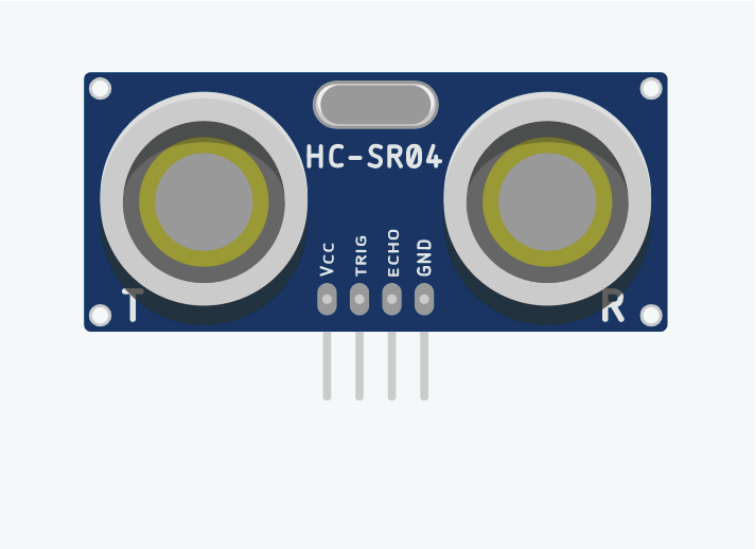
Test this with an IR sensor and it should work well.

Using whatever you have learnt till now, you will have to build a line following robot.
There will be a path made from black tape. Your robot should follow it till the end of the path. Make sure the sensors are sensitive enough to detect within the 1cm range.

At last, but not the least, we shall see the working of the ultrsonic sensor.

The ultrasonic sensor, as the name suggests, propagates ultrasonic sound waves and senses the echo. It figures the time taken for the wave to bounce of an object and return and this helps to calculate the distance between the sensor and an obstacle.

This is how a typical ultrasonic sensor looks.



The one labeled T is the transmitter and the one labeled R is the receiver. The Trig pin receives the command to trigger a pulse and then the Echo pin sends the time taken for the feedback.

For checking the time taken to receive the echo, we use the pulseIn command. pulseIn command takes two arguments, the pin number (of echo pin) and the state from which you want to measure the echo time. The output will be of the form of time in microseconds.

But now how to calculate distance? Well, we know the time it took and the speed of sound (approx. 343m/s). Using basic physics, we can calculate the distance as speed x time.
For precision, lets use units in mm and time in µs.
So speed = 0.343mm/µs

=> d = (t x 0.343)/2 mm
Divide it by 2 because it travels to and back (Two way trip!).

Connections->
VCC to 5V
GND to GND
TRIG to pin A
ECHO to pin B

Now lets see the structure of the code.

First we will need to set the pin A low. After a tiny delay, set it high for a brief period of time and then set it back to low. When you do so, read pulseIn into pin B, starting from high. And finally calculate the distance print the output using serial monitor. I will explain the serial monitor just after this code.

```
float t, d;

void setup()
{
  pinMode(5, OUTPUT);  // trig pin
  pinMode(6, INPUT);   // echo pin
  Serial.begin(9600);
  // communication rate = 9600 bits/s
}

void loop()
{
  // Send a pulse
  digitalWrite(5, LOW);
  delayMicroseconds(2);
  digitalWrite(5, HIGH);
  delayMicroseconds(10);
  digitalWrite(5, LOW);

  // See time taken
  t = pulseIn(6, HIGH);
  d = (0.343 * t)/2;

  Serial.println(d);   // Print output

  delay(100);
}
```

To see where the output is being printed, go to tools->Serial Monitor. Or click on the lens button towards the top right corner of the IDE.

Now that you are slightly familiar with the serial monitor, I'll just explain it briefly.

You can imagine the serial monitor as your output screen. We use it to read data which needs to be seen by the user.

Serial.begin() command states the rate of communication. We mostly use it at the rate of 9600bits/s.

Normally, we use 2 print commands->

```
Serial.print()
Serial.println()
```

Serial.print() prints the data within the same line.
Serial.println() prints the data in a new line.
This is really useful, specially to view analog values.

The serial monitor also helps us to take input from the user. There are two main commands for this.

```
Serial.available()
Serial.read()
```

Serial.available() sees if there is an input given by the user. We then use Serial.read() to take input characterwise. The value given by Serial.read() is the ASCII value of the character.

I suggest you to test these commands by yourself to understand the exact execution of the serial commands.

Finally you've reached the end of the basics of programming with arduino! Now your final task is... suspense... suspense... suspense...

Think of a project. An idea, that maybe hard to execute, that may need more manpower, that may need more knowledge, but you can do. It necessarily not be a robot. It could be a simple device, or a program that runs on different platforms.

To give you a little thought, your seniors have done an automatic water level sensor. They programmed it such that the water in the tank stops filling automatically when it nearly reaches the max level.

So you don't have to build a robot exactly. Just devices you could use to make the use of anything in real life easier.

Here are few links for your reference.

Video tutorials:-
https://youtu.be/zij_DuBjK5A
https://youtu.be/PsT6Xuwap58
https://youtu.be/ptSTmREkUso
https://youtu.be/YFkn4O98BG0

Arduino components:-
https://www.arduino.cc/reference/en