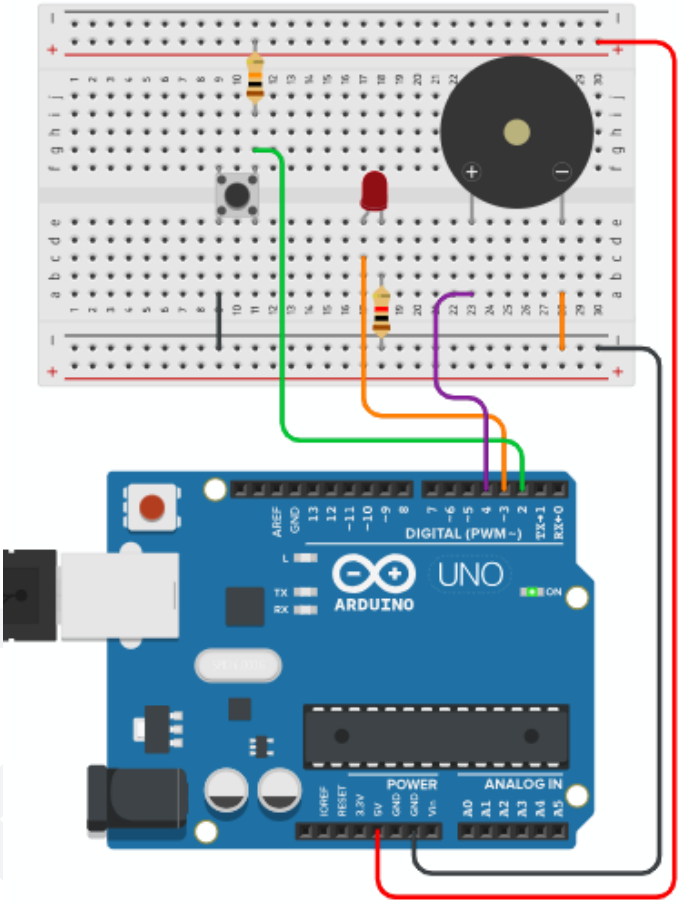
# Activity: Write an Arduino Program for Using the Serial Monitor

**Objective:**




In all the Arduino based activities so far, we have learnt how to control LEDs, Buzzer, multiple LEDs, interfacing of push Button switch etc but we have never used anything that gives output in the form of text or receives input in the form of text. Introducing the Arduino Serial Monitor utility. Here we can send output in the form of text and also receive inputs in the form of text. And text does not mean just alphabets. It can be alphabets (a-z and A-Z), single and multi digit numbers (9, 99, 999 etc.) and Strings ("robot", "sensor" etc.) basically any word within " ". But the Serial monitor is more than just a utility. It is a full fledged communication channel where data as well as commands can be sent and received.

For any kind of communication to happen there has to be a sender and a receiver. And the sender has to send data with the same speed with which the receiver can handle the data or else there will be Data Loss.

This communication speed is called "**Baud Rate**" in the world of Arduino programming. So, technically, Baud Rate is the predefined speed at which the sender and the receiver agree to communicate before the communication actually begins.

So, in today's activity we will learn how to use the Serial monitor for:

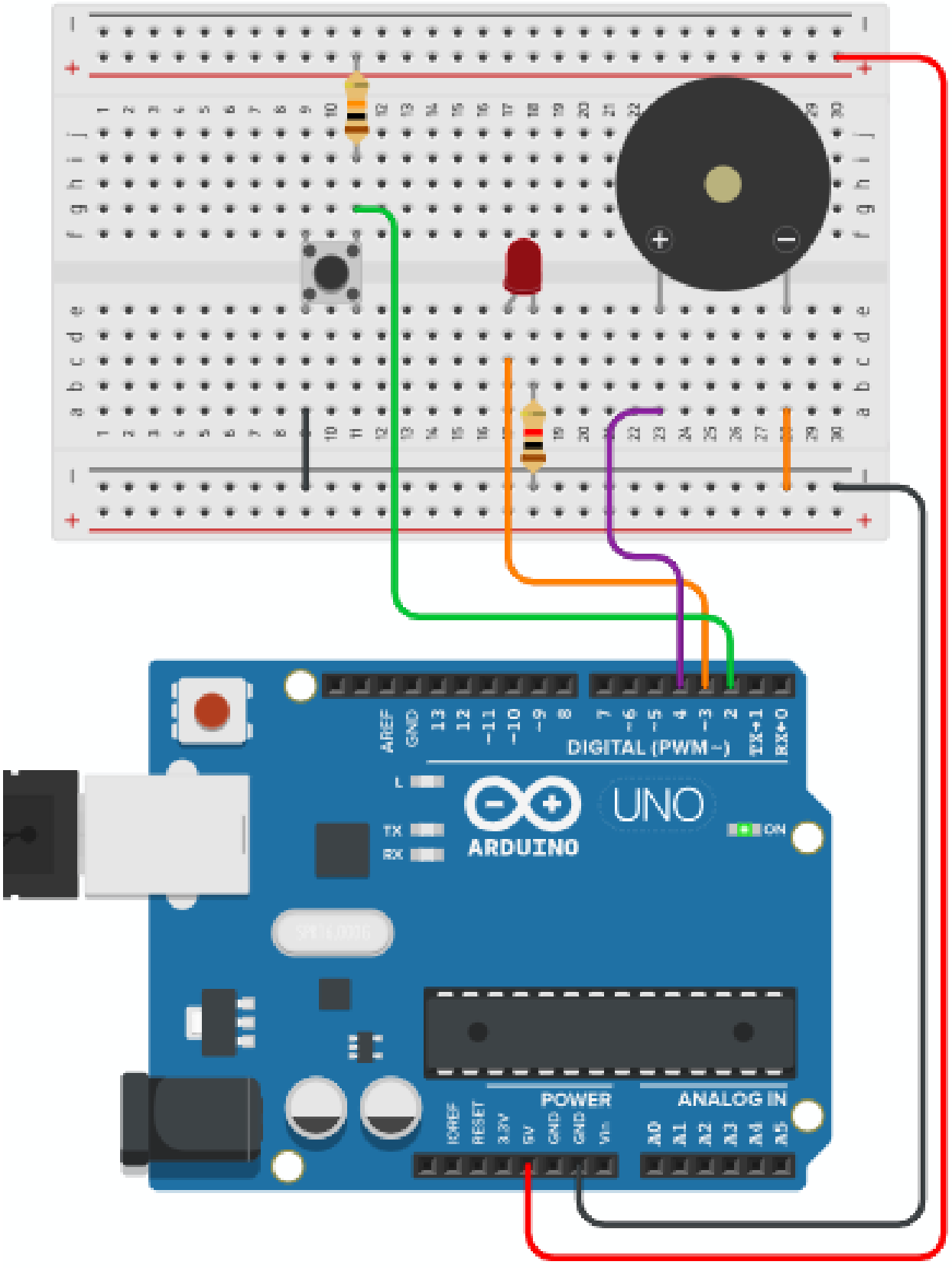
1. Sending alphabets, numbers and Strings to the Computer.
2. Receiving alphabets, numbers and String from the Computer.
3. Sending status of Button, LED and Buzzer from Arduino to Computer.
4. Receiving command from the Computer to control the LED and Buzzer.

**Materials Required:**

| S.no. | Name | Qty | Image |
|---|---|---|---|
| 1 | Arduino | 1 |  |
| 2 | Breadboard | 1 |  |
| 3 | M - M jumper wires | 10 |  |
| 5 | USB cable for Arduino Nano | 1 |  |

| | | | |
|---|---|---|---|
| 6 | Resistors:<br>10k (Brown Black Orange) | 1 |  |
| 7 | 5v Micro Buzzer | 1 |  |
| 8 | 4-pin tact switch | 1 |  |
| 9 | 5mm Red LED | 1 |  |
| 10 | 1k resistor | 1 |  |

**Connection Diagram:**



**Explanation:**

The above connection diagram shows the Red LED, the push Button and the Buzzer are all connected to the Arduino with the different values of resistors and connection wires. Although the overall circuit appears to be visually complex, it is quite easy to understand if we take one connection at a time. Here it is.

1. The Red LED is connected such that its +ve terminal is connected to the Arduino digital pin 3 (D3) and the -ve terminal is connected to one side of the 1k resistor. The other side of the 1k resistor is connected to the -ve of the breadboard.

2. The Buzzer is connected such that its +ve terminal is connected to the Arduino digital pin 4 (D4) and the -ve terminal is connected to the -ve of the breadboard.

3. The push Button is connected such that its top-right pin is connected to the +ve of the breadboard through a 10k resistor acting as a pull-up and the same pin is also connected to arduino digital pin 2 (D2). The bottom-left pin is connected to the -ve of the breadboard.

**Arduino Code:**

Using the Button as a Momentary input is something that we have already done earlier but we didn;t call it that back then. Remember we already controlled an LED and then a bunch of LEDs with a push Button. That was all Momentary behavior. The LEDs stayed On as long as the Button stays pressed and stayed Off as long as the Button stays released.

All we need to do here is use a digitalRead on the pin connected to the Button and then digitalWrite the inverse of its State to the LED and the Buzzer.

Here is the Arduino code.

```
#define Button 2              // defining pin 2 as "Button"
#define Red    3              // defining pin 3 as "Red"
#define Buzzer 4              // defining pin 4 as "Buzzer"


int Button_state;               // state variable for button
```

```
int Counter = 0;              // Counter variable for Toggle
String serial_data = "";      // variable for serial data


void setup()
{
 pinMode(Buzzer, OUTPUT);   //declaring "Buzzer" as output
 pinMode(Red, OUTPUT);        //declaring "Red" as output
 pinMode(Button, INPUT); //declaring "Button" as input
 Serial.begin(9600);      //declaring Baud Rate of 9600 bps
}


void loop()
{
  Button_state=digitalRead(Button); //check if button pressed

  if(Button_state == 0)                 // if button is pressed
  {
    Counter=Counter+1;

    Serial.print("Counter = ");
    Serial.print(Counter);
    Serial.println(" Button Pressed");

    delay(500);
  }

  else                          // if button is not pressed
  {
    Serial.print("Counter = ");
    Serial.print(Counter);
    Serial.println(" Button Released");
    delay(500);
  }



  if(Serial.available())   // if anything available on serial
  {
    serial_data=Serial.readString();  // store it in variable
```

```
    if(serial_data == "LED On")
    {
     digitalWrite(Red, HIGH);
    }

    if(serial_data == "LED Off")
    {
     digitalWrite(Red, LOW);
    }

    if(serial_data == "Buzzer On")
    {
     digitalWrite(Buzzer, HIGH);
    }

    if(serial_data == "Buzzer Off")
    {
     digitalWrite(Buzzer, LOW);
    }
  }
}
```

**Explanation:**

```
#define Button 2            // defining pin 2 as "Button"
#define Red    3            // defining pin 3 as "Red"
#define Buzzer 4            // defining pin 4 as "Buzzer"
```

The first three lines are simply defining which pins are going to be used as what. This makes it easier for the coder or user to refer to the pins in a more human understandable form. So, instead of referring to them using numbers, we give them names such as Buzzer, Red (LED) and Button.

```
int Button_state;               // state variable for button
int Counter = 0;              // Counter variable for Toggle
String serial_data = "";     // variable for serial data
```

In these lines we are declaring that we are going to use an Integer type variable with the name “Button_state”, an Integer type variable with the name “Counter” which is initialized with value 0 and a String type variable named serial_data which is initialized with a blank string.

```
void setup()
{
 pinMode(Buzzer, OUTPUT); //declaring "Buzzer" as output
 pinMode(Red, OUTPUT);    //declaring "Red" as output
 pinMode(Button, INPUT); //declaring "Button" as input
 Serial.begin(9600);     //declaring Baud Rate of 9600 bps
}
```

Here we are declaring that the “Buzzer” will be used as an Output, the Red (LED) will be used as an Output and that the “Button” will be used as an Input. Note that, here, we are using a new and different type of declaration as well. Here, we are declaring that we are going to begin serial communication at a speed of 9600 bits per second

```
  Button_state=digitalRead(Button); //check if button pressed
```

Here, we are first reading the state of the Push Button and storing it in the variable “Button_state”. Pressed is stored as a 0 and Released is stored as a 1.

```
  if(Button_state == 0)             // if button is pressed
  {
    Counter=Counter+1;   // increment counter value by 1

    Serial.print("Counter = ");
    Serial.print(Counter);
    Serial.println(" Button Pressed");

    delay(500);
  }
```

If the button is pressed, we increment the value of the counter variable by 1. Then display “Counter = ” on the Serial monitor followed by the value of that counter variable and finally display “ Button Pressed”. All these things are displayed in the same line on the serial monitor.

To send something to the serial monitor (Computer) we use “Serial.print() or Serial.println()” as shown in the above code segment. While “Serial.print()” displays the content and keeps the cursor on the same line position as the last character printed, the “Serial.println()” displays the content and shifts the cursor to the next line. That small “ln” is all the difference. It means newline

```
else                          // if button is not pressed
{
  Serial.print("Counter = ");
  Serial.print(Counter);
  Serial.println(" Button Released");
  delay(500);
}
```

These lines get executed when the button is not in the pressed state and that is why there is no counter increment but there is still a display of the counter value. Additionally, since we also want to display that the button is not pressed, we display that the button is released and then there is a small delay.

```
if(Serial.available())   // if anything available on serial
{
  serial_data=Serial.readString();  // store it in variable

  if(serial_data == "LED On")
  {
   digitalWrite(Red, HIGH);
  }

  if(serial_data == "LED Off")
  {
   digitalWrite(Red, LOW);
  }

  if(serial_data == "Buzzer On")
  {
```

```
     digitalWrite(Buzzer, HIGH);
    }

    if(serial_data == "Buzzer Off")
    {
     digitalWrite(Buzzer, LOW);
    }
  }
```

The first line checks if there is any data available for the Arduino to receive from the Computer (USB connection) Serial port. Then the received serial data is stored in the variable named “serial_data”.

Then a series of comparisons starts where the received serial data is compared against fixed words which we are using as commands to turn the Red LED and the Buzzer On and Off. Here are the word commands:

1. “LED On” for turning the Red LED On.
2. “LED Off” for turning the Red LED Off.
3. “Buzzer On” for turning the Buzzer On.
4. “Buzzer Off” for turning the Buzzer Off.

**Note:** “=” is used for assigning a value and “==” is used for comparing with a value

**Outcome and Observations:**

1. When powered on, the Red LED remains Off and the Buzzer remains Off as well.

2. Start the Arduino Serial Monitor with 9600 set as the Baud Rate.

3. We see that the Serial monitor is displaying the following continuously

```
Counter = 0 Button Released
Counter = 0 Button Released
Counter = 0 Button Released
```

4. Now, we press the button for sometime and then release it we observe the following

```
Counter = 1 Button Pressed
Counter = 2 Button Pressed
Counter = 0 Button Released
```

5. As long as the Button is pressed, the counter value keeps incrementing by 1 and “Button Pressed” is displayed.

6. As soon as the Button is released, the counter value becomes constant and “Button Released” is displayed.

7. When we type “LED On” in the text box in the serial monitor and press Enter, the Red LED turns On.

8. When we type “LED Off” in the text box in the serial monitor and press Enter, the Buzzer turns Off.

9. When we type “Buzzer On” in the text box in the serial monitor and press Enter, the Red LED turns On.

10. When we type “Buzzer Off” in the text box in the serial monitor and press Enter, the Buzzer turns Off.