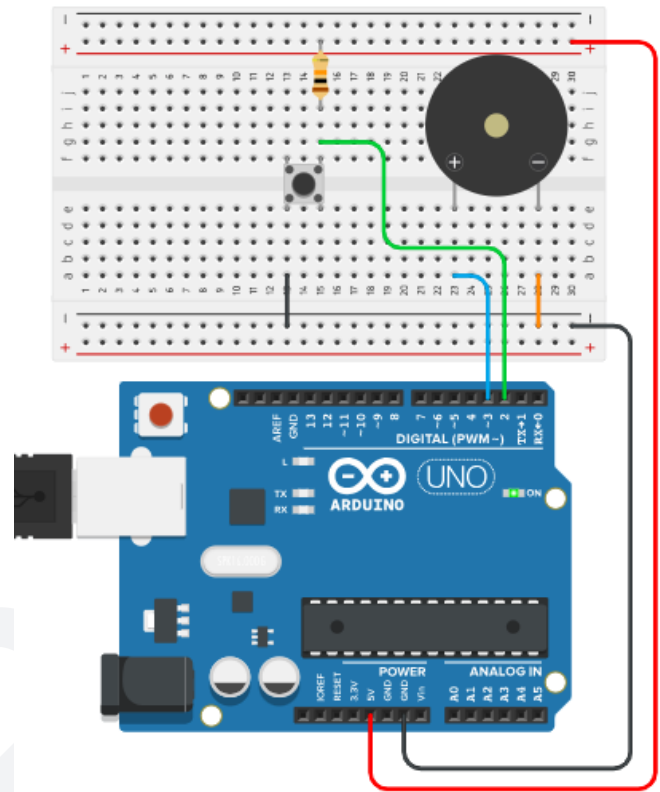


Activity: Generate Beeping Patterns using a Push Button and Buzzer

Objective:

So far in our activities we have already controlled single as well as multiple LEDs with the Arduino program. We have already seen how to control outputs using “digitalWrite”. But LEDs are not the only output devices that can be interfaced with the Arduino.

It is a well established fact that “Light”, “Sound”, and “Motion” are the three best tools to impart knowledge in a way that it stays with students. And so far we have dealt with only Light in the form of different colored LEDs. But digital systems don't just have LEDs as Outputs, they have other types of Outputs as well and so we need to learn how to interface these Output devices with the Arduino.






One of the most commonly used output devices is the Buzzer. A Buzzer is a simple electronic component which produces a continuous sound output when powered up. You can find a buzzer inside alarm clocks, set-top boxes, microwave ovens, washing machines, mobile phones, various alarms and security systems etc. A simple buzzer can do a lot more than producing a continuous sound output. It can be modified to produce different beeping patterns so that the listener can understand the type of notification depending on the pattern without even going in front of the system.

So, in today's activity we will first learn how to connect a Buzzer to the Arduino, then we will learn how to control this Buzzer using the “press” and “release” operation of a push button and finally, how to use the button to generate different beep patterns using the button and the buzzer. This activity will be divided into 2 parts.

1. Controlling the Buzzer with a push button.
2. Producing different Beep patterns based on a counter value which will be changed every time the button is pressed.

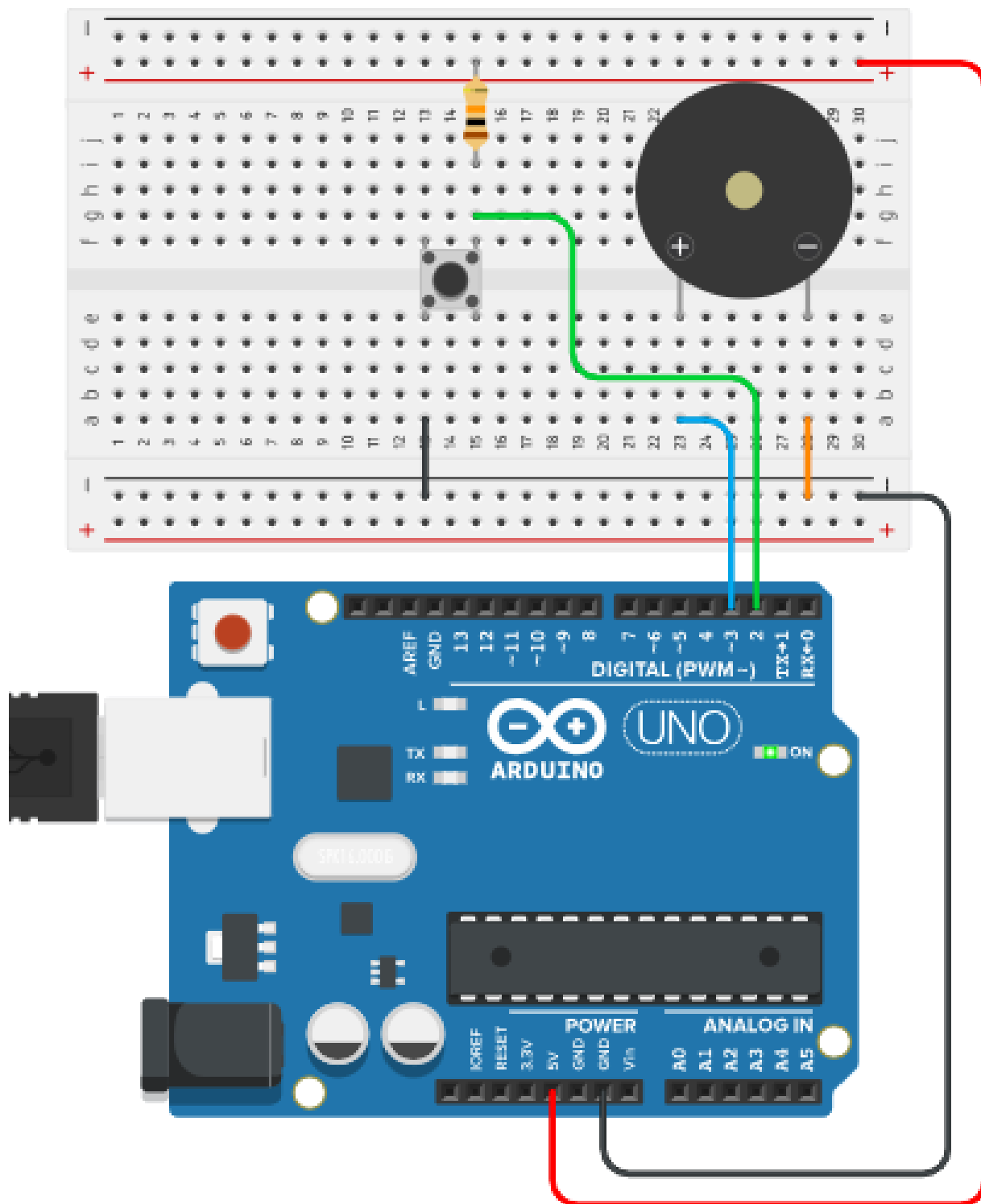
Materials Required:

S.no.	Name	Qty	Image
1	Arduino	1	 A blue Arduino Uno R3 microcontroller board. It features a USB Type-B port, a DC power jack, a reset button, and a large integrated circuit (ATmega328P) in the center. The board is populated with various components like capacitors and resistors.
2	Breadboard	1	 A standard white breadboard used for prototyping electronics. It has a grid of holes for components and a central strip of holes for connecting components across the board.
3	M - M jumper wires	10	 Ten jumper wires of various colors (red, yellow, green, blue, black) with male-to-male (M-M) connectors on both ends. They are arranged in a fan shape, showing the different colors and the metal pins at the ends.

5	USB cable for Arduino	1	
6	Resistors: 10k (Brown Black Orange)	1	
7	5v Micro Buzzer	1	
8	4-pin tact switch	1	

Part 1:

Connection Diagram:



Explanation:

Looking at the connection diagram shown above, some things are immediately evident:

1. A 10k resistor is connected as pull-up between the +ve of the breadboard and the top-right pin of the push button as shown in the above diagram.
2. One side of the Push button switch is connected to Digital Pin 2 (D2) of the Arduino while the other side is connected to -ve of the Breadboard with a connection wire (shown with the black line just below the switch).
3. The +ve and -ve power pins of the Arduino are connected to the +ve and -ve lines of the Breadboard.
4. Pressing the Button connects Pin 2 (D2) of the Arduino to Gnd (-ve power) and releasing the button leaves the Pin 5 connected to 5v through the 10k resistor acting as an external pull-up.
5. The Buzzer is connected such that its +ve pin is connected to Digital Pin 3 (D3) of the Arduino with a connection wire (shown in) while its -ve pin is connected to the -ve of the Breadboard with a connection wire ()

Arduino Code:

LEDs give Light as output and we have already learnt how to control single as well as multiple LEDs using Arduino code as well as a Push Button. Now, we move on to produce sound output using a Buzzer.

There is absolutely no difference whatsoever in the logic that turns an LED on-off or a Buzzer on-off. All we need to do is specify the Arduino pin to which the buzzer is connected and then use the “digitalWrite(pin, state);” function to turn the Buzzer On or Off using HIGH or LOW inside the digitalWrite.

Additionally, we need to keep checking continuously whether the Button is in the Pressed state or Released state. We want the Buzzer to be On when the Push Button is in its Pressed state and Off when the Push Button is in its Released state. Notice that Pressing the Button makes the pin LOW while turning On the Buzzer makes the pin HIGH. So the relation between the input and output is Opposite.

So, the logic of Buzzer On is inverse of the logic of Button On.

This will be more clear when you go through the code below.

Here is the Arduino code.

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

int Button_state;

void setup()
{
  pinMode(Buzzer, OUTPUT); //declaring "Buzzer" as output
  pinMode(Button, INPUT);  //declaring "Button" as input
}

void loop()
{
  Button_state=digitalRead(Button); //check if button pressed
  digitalWrite(Buzzer, !Button_state); // turn On the Buzzer
}
```

Explanation:

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

The first two 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 and Button.

```
int Button_state;
```

In this line we are declaring that we are going to use an Integer type variable with the name "Button_state". A variable can be of many types depending on the value we want it to hold. It can be an Integer (number without a decimal point), a Character (a-z or A-Z), a String (set of alphabets e.g. "robot" or "ROBOT" or "Robot") etc.

```
void setup()
{
  pinMode(Buzzer, OUTPUT);      //declaring "Buzzer" as output
  pinMode(Button, INPUT);      //declaring "Button" as input
}
```

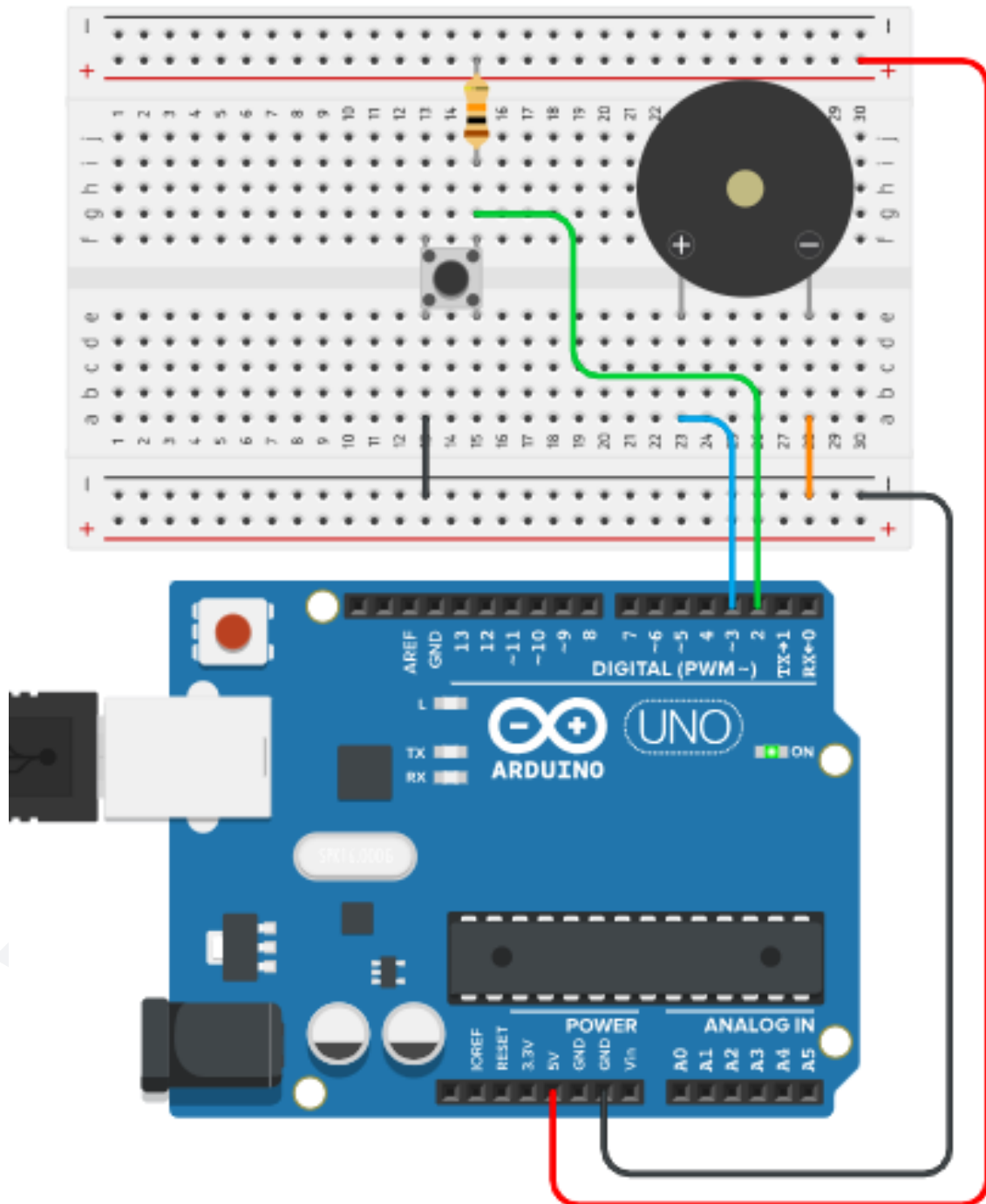
Here we are declaring that the "Buzzer" will be used as an Output and that the "Button" will be used as an Input. Note that, here, we are using "INPUT" instead of "INPUT_PULLUP". This is because Pressing the Button makes D2 LOW and Releasing it makes D2 HIGH. This default HIGH state for pin D2 is provided here by the external 10k resistor acting as Pull-Up.

```
void loop()
{
  Button_state = digitalRead(Button);
  digitalWrite(LED, !Button_state);
}
```

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. And then we are turning the LED On or Off by writing this 0 or 1 on the "LED" pin i.e. D13.

Part 2:

Connection Diagram: (connections remain unchanged from part 1)



Explanation:

In the second part of the activity, there is no change in the circuit components or its connections. Take a look at the above connection diagram and you will observe the following:

1. A 10k resistor is connected as pull-up between the +ve of the breadboard and the top-right pin of the push button as shown in the above diagram.
2. One side of the Push button switch is connected to Digital Pin 2 (D2) of the Arduino while the other side is connected to -ve of the Breadboard with a connection wire (shown with the black line just below the switch).
3. The +ve and -ve power pins of the Arduino are connected to the +ve and -ve lines of the Breadboard.
4. Pressing the Button connects Pin 2 (D2) of the Arduino to Gnd (-ve power) and releasing the button leaves the Pin 5 connected to 5v through the 10k resistor acting as an external pull-up.
5. The Buzzer is connected such that its +ve pin is connected to Digital Pin 3 (D3) of the Arduino with a connection wire (shown in) while its -ve pin is connected to the -ve of the Breadboard with a connection wire ()

Arduino Code:

Now that we have already understood how a push button works for Buzzer, we move on to generate different Beeping patterns using the same Buzzer and Button setup .

The logic is not exactly the same as LEDs but quite similar. We take an integer type Counter variable and increment it every time the button is pressed. And depending on the present Value of the Counter variable we play different Beeping Patterns.

Here, we use a new technique called “functions”. A function is a group of statements that get executed whenever the name of the function is called. A function is written separately outside the ”setup” or “loop” sections. All this will be much clearer once we go through the code.

Here is the code for generating different beeping patterns with a push button switch.

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

int Button_state;       // variable to store Button status
int Counter=0;         // variable to store Counter value

void Beep_pattern1()   // first beeping pattern
{
    digitalWrite(Buzzer, HIGH); // Buzzer turned On
    delay(50);                // wait for 50ms
    digitalWrite(Buzzer, LOW); // Buzzer turned OFF
    delay(50);                // wait for 50ms

    digitalWrite(Buzzer, HIGH); // Buzzer turned On
    delay(50);                // wait for 50ms
    digitalWrite(Buzzer, LOW); // Buzzer turned OFF
    delay(50);                // wait for 50ms

    digitalWrite(Buzzer, HIGH); // Buzzer turned On
    delay(50);                // wait for 50ms
    digitalWrite(Buzzer, LOW); // Buzzer turned OFF
    delay(50);                // wait for 50ms

    digitalWrite(Buzzer, HIGH); // Buzzer turned On
    delay(50);                // wait for 50ms
    digitalWrite(Buzzer, LOW); // Buzzer turned OFF
    delay(50);                // wait for 50ms
}

void Beep_pattern2()   // 2nd beeping pattern
{
    digitalWrite(Buzzer, HIGH);
    delay(20);
    digitalWrite(Buzzer, LOW);
    delay(50);

    digitalWrite(Buzzer, HIGH);
    delay(30);
    digitalWrite(Buzzer, LOW);
    delay(30);
}
```

```
digitalWrite(Buzzer, HIGH);
delay(30);
digitalWrite(Buzzer, LOW);
delay(30);

digitalWrite(Buzzer, HIGH);
delay(30);
digitalWrite(Buzzer, LOW);
delay(30);
}
```

```
void Beep_pattern3() // 3rd beeping pattern
{
digitalWrite(Buzzer, HIGH);
delay(30);
digitalWrite(Buzzer, LOW);
delay(50);

digitalWrite(Buzzer, HIGH);
delay(30);
digitalWrite(Buzzer, LOW);
delay(50);

digitalWrite(Buzzer, HIGH);
delay(400);
digitalWrite(Buzzer, LOW);
delay(50);

digitalWrite(Buzzer, HIGH);
delay(30);
digitalWrite(Buzzer, LOW);
delay(50);

digitalWrite(Buzzer, HIGH);
delay(30);
digitalWrite(Buzzer, LOW);
delay(50);
}
```

```
void Beep_pattern4() // 4th beeping pattern
{
```

```

digitalWrite(Buzzer, HIGH);
delay(200);
digitalWrite(Buzzer, LOW);
delay(50);

digitalWrite(Buzzer, HIGH);
delay(50);
digitalWrite(Buzzer, LOW);
delay(50);

digitalWrite(Buzzer, HIGH);
delay(50);
digitalWrite(Buzzer, LOW);
delay(50);

digitalWrite(Buzzer, HIGH);
delay(50);
digitalWrite(Buzzer, LOW);
delay(50);
}

void setup()
{
  pinMode(Buzzer, OUTPUT); // declaring "Buzzer" as output
  pinMode(Button, INPUT); // declaring "Button" as input
}

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

  if(Button_state == LOW) // if button is in pressed state
  {
    Counter = Counter + 1; // increment Counter by 1
    delay(500); // wait for half a second
  }

  if(Counter == 1) // if Counter value is 1
  {
    Beep_pattern1(); // play 1st pattern
    delay(1000); // wait for a second
  }
}

```

```
}

if(Counter == 2)           // if Counter value is 2
{
    Beep_pattern2();      // play 2nd pattern
    delay(1000);          // wait for a second
}

if(Counter == 3)           // if Counter value is 3
{
    Beep_pattern3();      // play 3rd pattern
    delay(1000);          // wait for a second
}

if(Counter == 4)           // if Counter value is 4
{
    Beep_pattern4();      // play 4th pattern
    delay(1000);          // wait for a second
}

if(Counter == 5)           // if Counter value is 5
{
    Counter = 0;          // reset Counter to 0
}
}
```

Explanation:

The above code does the following:

1. Check whether the push button is pressed or not.
2. If the button is pressed, increment a Counter value by 1.
3. If the Counter value is 1, play beeping pattern 1 on the Buzzer.
4. If the Counter value is 2, play beeping pattern 2 on the Buzzer.
5. If the Counter value is 3, play beeping pattern 3 on the Buzzer.
6. If the Counter value is 4, play beeping pattern 4 on the Buzzer.
7. If the Counter value is 5, reset the Counter value back to 0.

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

These lines define that the Buzzer and the Button are connected to Arduino digital pins 3 and 2 respectively.

```
int Button_state;         // variable to store Button status  
int Counter=0;           // variable to store Counter value
```

Here, we have taken 2 variables, both integer types, one for storing the button status and the other for storing the Counter value. The Counter value has been initialized by 0.

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

This is the line which actually checks the state of the button whether it is in the pressed state or released state and then stores that state in the "Button_state" variable. The way we have connected our push button, "digitalRead" of a pressed state will be 0 (LOW) while "digitalRead" of a released state will be 1 (HIGH).

```
if(Button_state == LOW) // if button is in pressed state  
{  
    Counter = Counter + 1; // increment Counter by 1  
    delay(500);           // wait for half a second  
}
```

These lines are first comparing if the "Button_state" variable has detected a "LOW" (a pressed state). And if this is true, then increment the Counter variable by 1 (which was initially 0) and then wait for half a second (500ms).

```
void Beep_pattern1() // first beeping pattern  
{  
    digitalWrite(Buzzer, HIGH); // Buzzer turned On  
    delay(50);                  // wait for 50ms  
    digitalWrite(Buzzer, LOW); // Buzzer turned OFF  
    delay(50);                  // wait for 50ms  
  
    digitalWrite(Buzzer, HIGH); // Buzzer turned On  
    delay(50);                  // wait for 50ms  
    digitalWrite(Buzzer, LOW); // Buzzer turned OFF
```

```

delay(50); // wait for 50ms

digitalWrite(Buzzer, HIGH); // Buzzer turned On
delay(50); // wait for 50ms
digitalWrite(Buzzer, LOW); // Buzzer turned OFF
delay(50); // wait for 50ms

digitalWrite(Buzzer, HIGH); // Buzzer turned On
delay(50); // wait for 50ms
digitalWrite(Buzzer, LOW); // Buzzer turned OFF
delay(50); // wait for 50ms
}

```

Here, the buzzer is being turned On and Off with different delay durations and this is what generates the particular beeping pattern. We are using 4 different patterns and using them as functions “Beep_pattern1()”, “Beep_pattern2()”, “Beep_pattern3()” and “Beep_pattern4()”.

Note that throughout the program, we have used “=” for assigning a value to a variable while “==” has been used for comparison and the comparison is always inside the braces of an “if” statement such as:

```

if(Counter == 3)

```

Here, we are comparing if the value of the variable “Counter” is equal to 3 or not. If it is 3, this “if” gives output as True. If, however, it is not 3, this “if” gives output as False.

```

Counter = 0;

```

Here, we are assigning the variable “Counter” the value 0. There is no necessity to stop at 3 or 4 beeping patterns. The integer variable “Counter” can very well accommodate values up to 65535. It is, however

Outcome and Observations:

1. For Part 1, once you are done compiling and uploading the code to the Arduino, you will observe the following:
 - a. When you press the Button and keep it pressed, the Buzzer turns On and Stays On as long as the Button remains pressed.
 - b. When you release the Button and it stays released, the Buzzer turns Off and Stays Off as long as the Button remains released.
 - c. No matter how fast or slow you press the Button, the Buzzer stays On when the Button is pressed and stays Off when the Button is released.

This type of behavior is called “Momentary” because the state of the output (the Buzzer) entirely depends on the state of the Input (the push button). Button On makes Buzzer On and Button Off makes Buzzer Off.

2. For Part 2, once you are done compiling and uploading the code to the Arduino, you will observe the following:
 - a. Initially, the Buzzer remains Off.
 - b. When the push button is pressed for the first time, the Counter value increments from 0 to 1 and plays the 1st beeping pattern on the Buzzer.
 - c. When the push button is pressed for the second time, the Counter value increments from 1 to 2 and plays the 2nd beeping pattern on the Buzzer.
 - d. When the push button is pressed for the third time, the Counter value increments from 2 to 3 and plays the 3rd beeping pattern on the Buzzer.
 - e. When the push button is pressed for the fourth time, the Counter value increments from 3 to 4 and plays the 4th beeping pattern on the Buzzer.
 - f. When the push button is pressed for the fifth time, the Counter value increments from 4 to 5 and the Counter value is re-initialized to 0. This operation readies the entire setup to start over again.