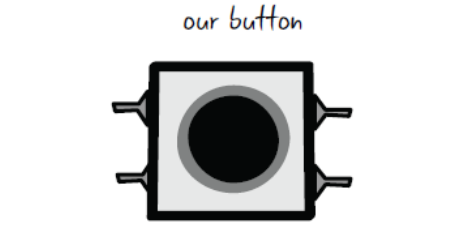
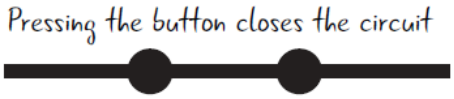
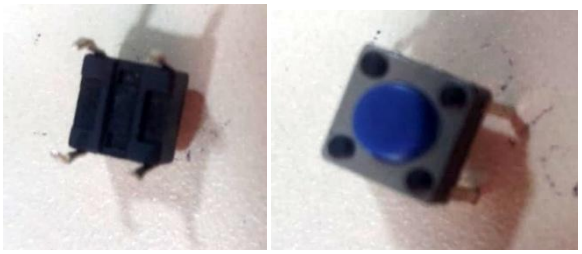
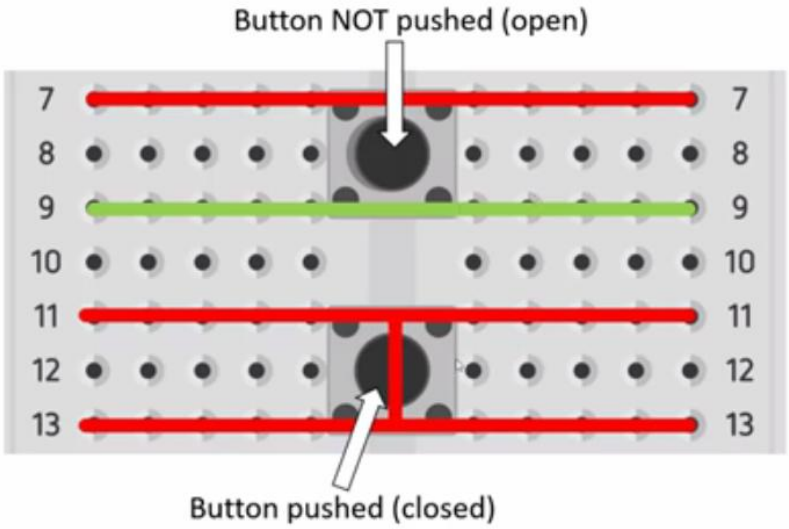
Experiment 5

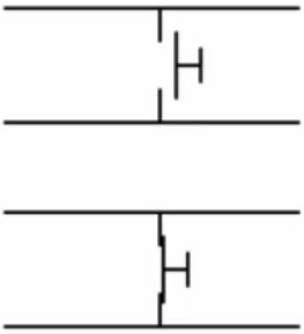
how to wire a pushbutton with four terminals (1a, 1b, 2a, 2b) and explain how it works step-by-step.
pushbutton terminal 1a and 1b, 2a and 2b











Pushbutton Terminology

A pushbutton with four terminals is typically a **SPST (Single Pole Single Throw) Pushbutton** with two sets of terminals. Here's what the terminals represent:

- **1a and 1b**: One pair of terminals (often connected internally).

- **2a and 2b**: Another pair of terminals (often connected internally).

When the button is not pressed:

- **1a** is **not connected** to **1b**.
- **2a** is **not connected** to **2b**.

When the button is pressed:

- **1a connects** to **1b**.
- **2a connects** to **2b**.

An SPST pushbutton is a very straightforward type of switch. It controls one circuit and has one action (making or breaking the connection) that happens when you press the button. An SPST pushbutton is a simple switch with just one button that either makes or breaks a connection when pressed. It has two states: on (when pressed) and off (when released).

Single Pole: This means the pushbutton controls only one circuit. There's just one wire that it can connect or disconnect.

Single Throw: This means the pushbutton has only one function: it can either make the circuit complete (turn something on) or break the circuit (turn something off). There's no extra setting—just one position when pressed and one when not pressed.

Wiring the Pushbutton

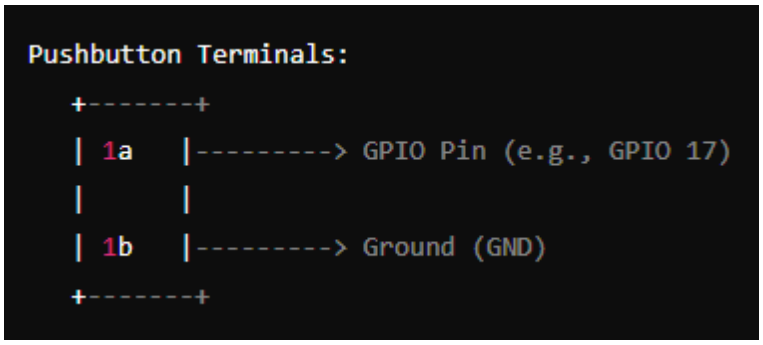
1. **Identify the Terminals**:
    - Look at your pushbutton and locate the terminals. They should be labeled 1a, 1b, 2a, and 2b.
2. **Determine Which Terminals to Use**:
    - For a basic circuit, you generally use only one pair of terminals. For simplicity, we'll use **1a** and **1b** for this example.
3. **Connecting the Pushbutton**:
    - **1a**: Connect to a GPIO pin on the Raspberry Pi (for example, GPIO 17).

- **1b**: Connect to Ground (GND) on the Raspberry Pi.

4. **Testing the Pushbutton**:
   - When the button is not pressed, the circuit between 1a and 1b is open (not connected).
   - When you press the button, 1a and 1b become connected, allowing current to flow through.

Diagram

Here's a simple diagram showing the connections:



SPST (Single Pole Single Throw) is a type of switch with one circuit path and two states: ON (connected) and OFF (disconnected).It’s used to control the flow of current in a single circuit.

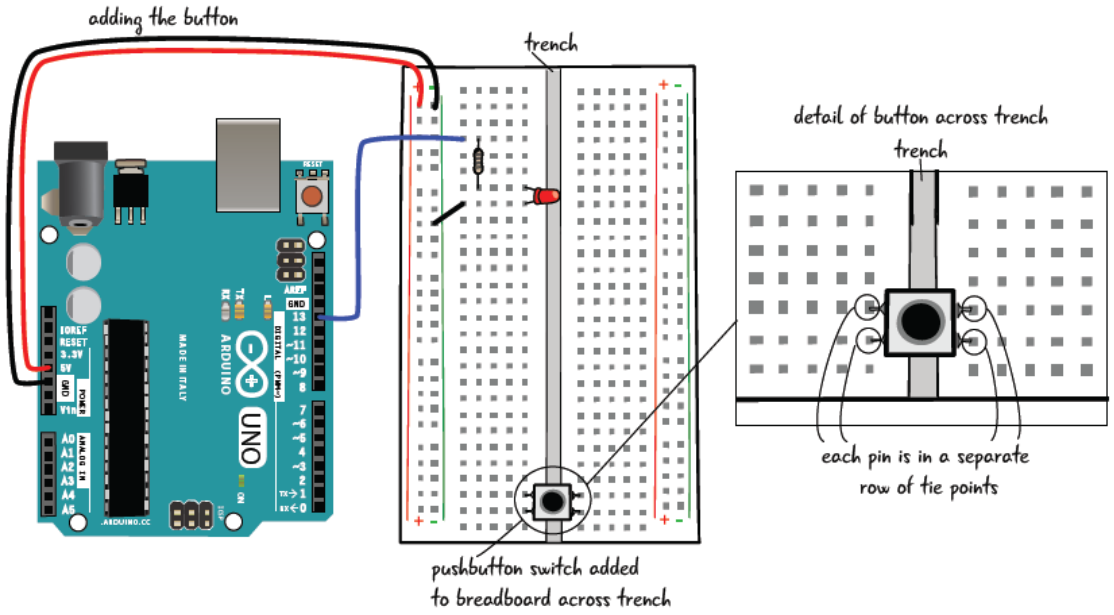


FIGURE 6.9: Adding the button to your breadboard

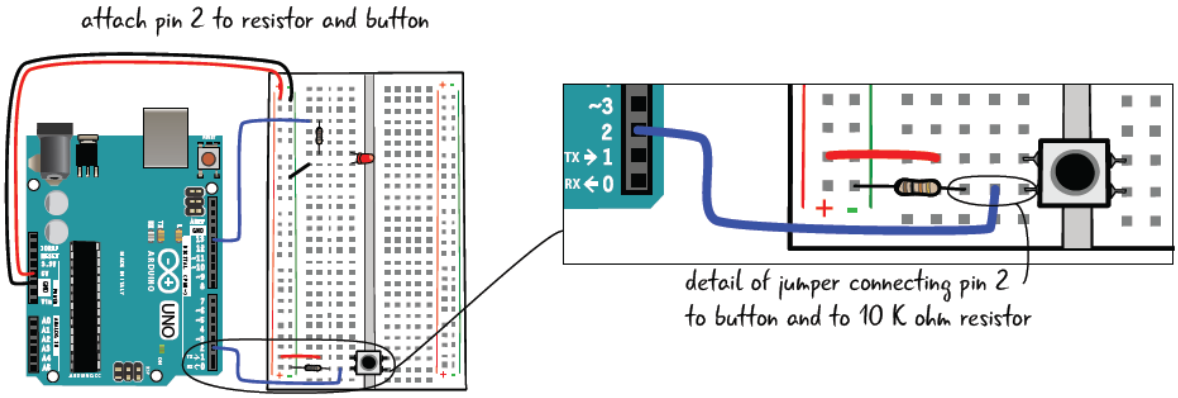


FIGURE 6.12: Adding the jumper to the digital pin

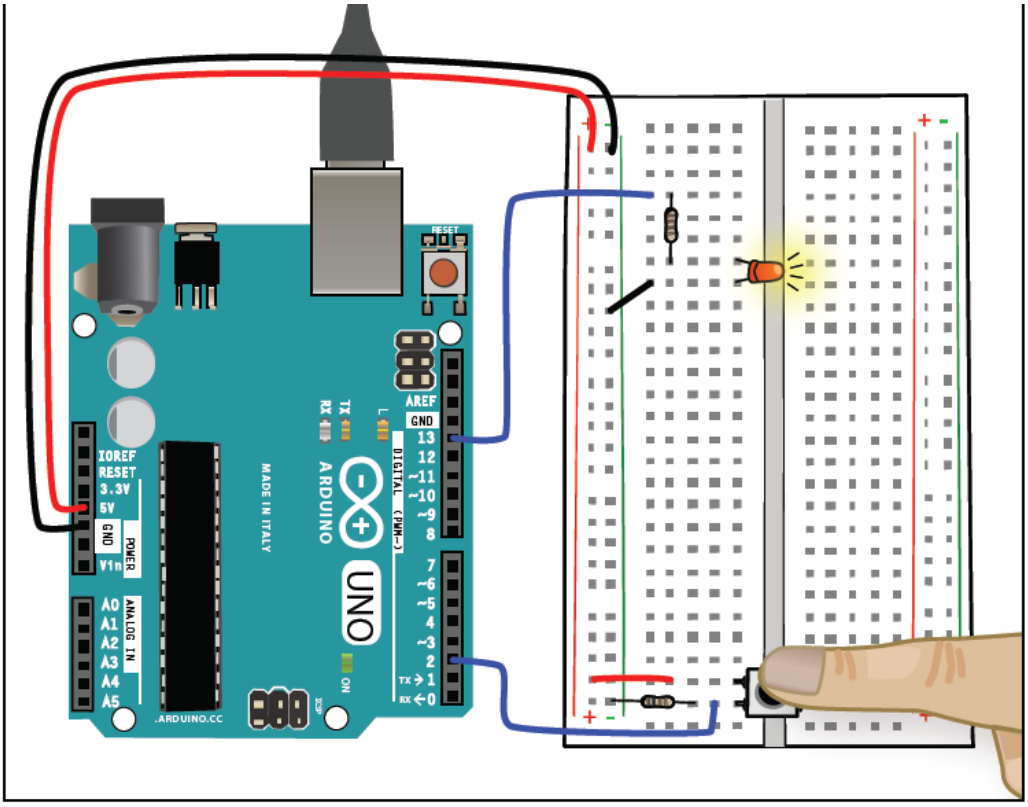
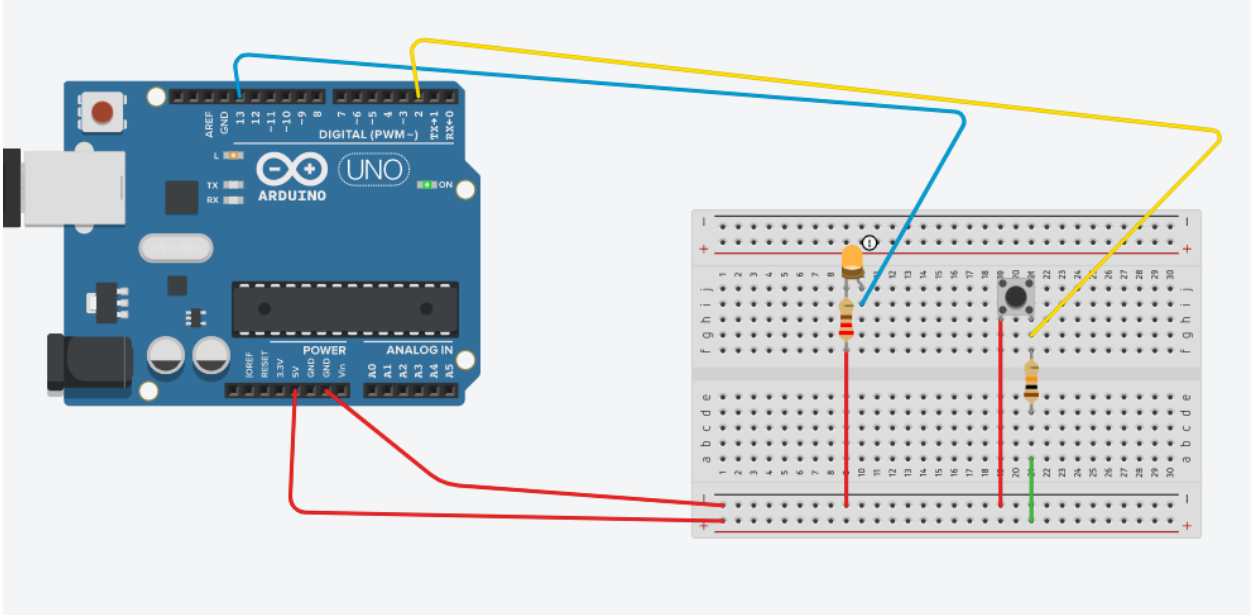


FIGURE 6.14: Press the button and the LED lights up.

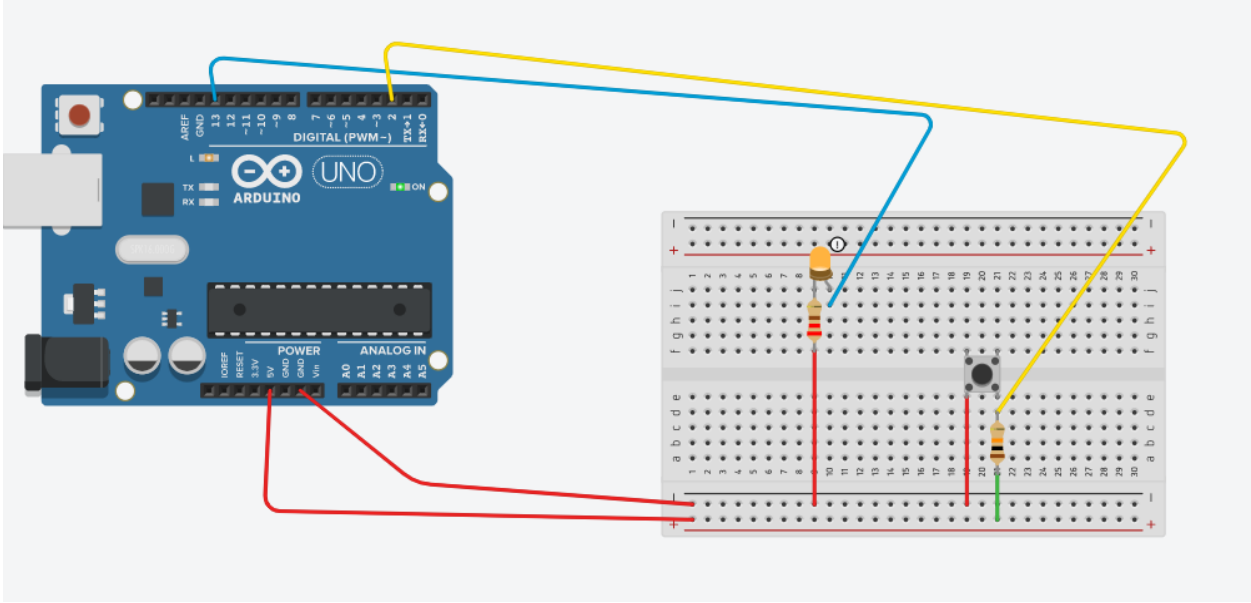
write a program to demonstrate LED with button

Press Pushbutton led will off

The pushbutton is not arranged properly, but it is still working because all its pins are in the same position, connecting the two terminals in the same line. its arrangement is not ideal because it connects terminals inappropriately.



The proper way to arrange the pushbutton is to place it across the middle gap of the breadboard or into the Dual In-line Package (DIP) socket. This ensures that the pins do not connect to the same terminal.



1. Set Up Your Circuit

1. **Create a New Circuit:**
   - Go to Tinkercad, and sign in.
   - Click on “Circuits” in the left menu.
   - Click the “Create new Circuit” button.
2. **Add Components:**
   - **Arduino Uno**: Search for "Arduino Uno" in the components menu and drag it onto the workspace.
   - **Pushbutton**: Search for "Pushbutton" and drag it onto the workspace.
   - **LED**: Search for "LED" and place it on the workspace.
   - **Resistors**: Search for "Resistor" and add two resistors (one for the LED and one for the button).
3. **Wire the Pushbutton:**
   - Place the pushbutton on the breadboard.
   - Connect one side of the pushbutton to digital pin 2 on the Arduino Uno.
   - Connect the other side of the pushbutton to the ground (GND) on the Arduino Uno.
   - Add a pull-down resistor (10kΩ) between the GPIO pin (digital pin 2) and ground. This ensures a stable LOW state when the button is not pressed.
4. **Wire the LED:**
   - Place the LED on the breadboard.
   - Connect the positive (long) leg of the LED to digital pin 13 on the Arduino Uno through a current-limiting resistor (220Ω or 330Ω).
   - Connect the negative (short) leg of the LED to the ground (GND) on the Arduino Uno.
5. **Check Connections:**
   - Ensure that all components are properly connected according to the wiring described.

2. Write the Code

1. **Open the Code Editor:**
    - Click on the "Code" button in the upper right corner of the Tinkercad interface.
    - Choose "Blocks + Text" mode to see both block-based and text-based code.
2. **Write the Arduino Code:**
    - Replace the default code with the following code to invert the LED behavior based on the pushbutton state:

Code 1: press pushbutton down to off LED

```
const int buttonPin = 2;    // the number of the pushbutton pin
const int ledPin =  13;     // the number of the LED pin

int buttonState = 0;        // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input with internal pull-up resistor:
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is LOW:
  if (buttonState == LOW) {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  } else {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
}
```

Code 2: press pushbutton down to on LED

```
const int buttonPin = 2;    // the number of the pushbutton pin

const int ledPin =  13;     // the number of the LED pin
```

```
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // Initialize the LED pin as an output
  pinMode(ledPin, OUTPUT);
  // Initialize the pushbutton pin as an input with internal pull-up resistor
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  // Read the state of the pushbutton value
  buttonState = digitalRead(buttonPin);
  // Check if the pushbutton is pressed.
  // If it is pressed, the buttonState is LOW
  if (buttonState == LOW) {
    // Turn LED on
    digitalWrite(ledPin, HIGH);
  } else {
    // Turn LED off
    digitalWrite(ledPin, LOW);
  }
}
```

Explanation

Constants and Variables

1. **const int buttonPin = 2;**
    - Defines the pin number for the pushbutton (digital pin 2).
2. **const int ledPin = 13;**
    - Defines the pin number for the LED (digital pin 13).
3. **int buttonState = 0;**
    - A variable to store the current state of the pushbutton (HIGH or LOW).

Setup Function

1. **void setup() { ... }**
    - Runs once when the Arduino is powered on or reset.
2. **pinMode(ledPin, OUTPUT);**
    - Configures ledPin (pin 13) as an output. This pin will control the LED.
3. **pinMode(buttonPin, INPUT_PULLUP);**
    - Configures buttonPin (pin 2) as an input with an internal pull-up resistor. This means the pin will read HIGH when the button is not pressed and LOW when the button is pressed.

Loop Function

1. **void loop() { ... }**
    - Continuously runs after setup().
2. **buttonState = digitalRead(buttonPin);**
    - Reads the current state of the pushbutton and stores it in buttonState. The value will be HIGH if the button is not pressed and LOW if it is pressed.
3. **if (buttonState == LOW) { ... } else { ... }**
    - Checks if the button is pressed (buttonState == LOW).
4. **digitalWrite(ledPin, LOW);**
    - Turns the LED off when the button is pressed.
5. **digitalWrite(ledPin, HIGH);**
    - Turns the LED on when the button is not pressed.

Summary

- **When the Button is Pressed**: The button pin reads LOW, and the LED turns off.
- **When the Button is Not Pressed**: The button pin reads HIGH, and the LED turns on.

Using raspberry pi Python code

To control an LED using a pushbutton with a Raspberry Pi, you can use Python with the RPi.GPIO library.

Hardware Setup

1. **Components Needed:**
    - Raspberry Pi (any model with GPIO pins)
    - Pushbutton
    - LED
    - 220Ω Resistor (for the LED)
    - 10kΩ Resistor (for the pushbutton)
    - Breadboard and jumper wires
2. **Connections:**
    - **Pushbutton:**
        - One side to GPIO pin (e.g., GPIO 17).
        - Other side to Ground (GND).
        - A 10kΩ resistor between the GPIO pin and 3.3V to pull-up.
    - **LED:**
        - Positive (long) leg to GPIO pin (e.g., GPIO 27) through a 220Ω resistor.
        - Negative (short) leg to Ground (GND).

Python Code

1. **Install RPi.GPIO Library:** If you haven't installed it yet, you can install it via pip:

    sudo apt-get update

sudo apt-get install python3-rpi.gpio

2. **Python Code:**

Here's a Python script to control the LED based on the pushbutton state:

```
import RPi.GPIO as GPIO
import time

# GPIO pin numbers
BUTTON_PIN = 17
LED_PIN = 27

# Set up GPIO mode
GPIO.setmode(GPIO.BCM)  # Use Broadcom pin numbering
GPIO.setup(LED_PIN, GPIO.OUT)  # Set LED pin as an output
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set button pin as an input with internal pull-up resistor

try:
    while True:
        # Read the state of the pushbutton
        button_state = GPIO.input(BUTTON_PIN)

        # Check if the pushbutton is pressed (LOW when pressed)
        if button_state == GPIO.LOW:
            # Turn LED on
            GPIO.output(LED_PIN, GPIO.HIGH)
        else:
            # Turn LED off
            GPIO.output(LED_PIN, GPIO.LOW)
```

```
        # Short delay to debounce button press
        time.sleep(0.01)

    except KeyboardInterrupt:
        # Clean up GPIO on Ctrl+C exit
        GPIO.cleanup()
```

Explanation of the Code

- **import RPi.GPIO as GPIO**: Imports the GPIO library to control the GPIO pins.
- **import time**: Imports the time library to add delays.
- **GPIO.setmode(GPIO.BCM)**: Sets the GPIO mode to use Broadcom pin numbers.
- **GPIO.setup(LED_PIN, GPIO.OUT)**: Configures the LED pin as an output.
- **GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)**: Configures the button pin as an input with an internal pull-up resistor.
- **button_state = GPIO.input(BUTTON_PIN)**: Reads the state of the pushbutton. The state will be GPIO.LOW when pressed and GPIO.HIGH when not pressed.
- **GPIO.output(LED_PIN, GPIO.HIGH)**: Turns the LED on when the button is pressed.
- **GPIO.output(LED_PIN, GPIO.LOW)**: Turns the LED off when the button is not pressed.
- **time.sleep(0.01)**: A small delay to debounce the button press.
- **GPIO.cleanup()**: Cleans up the GPIO settings on exit to avoid warnings.

Running the Code

1. Save the Python script (e.g., led_button.py).
2. Run it from the terminal:

```
python3 led_button.py
```

This script will keep running until you stop it with Ctrl+C. The LED will turn on when the pushbutton is pressed and turn off when the button is not pressed.

Inverted Python Code

If you want the LED to turn **off** when the pushbutton is pressed (and turn **on** when the button is not pressed), you'll need to invert the logic in the Python code.

```
import RPi.GPIO as GPIO
import time

# GPIO pin numbers
BUTTON_PIN = 17
LED_PIN = 27

# Set up GPIO mode
GPIO.setmode(GPIO.BCM)  # Use Broadcom pin numbering
GPIO.setup(LED_PIN, GPIO.OUT)  # Set LED pin as an output
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set button pin as an input with internal pull-up resistor

try:
    while True:
        # Read the state of the pushbutton
        button_state = GPIO.input(BUTTON_PIN)

        # Check if the pushbutton is pressed (LOW when pressed)
        if button_state == GPIO.LOW:
            # Turn LED off
            GPIO.output(LED_PIN, GPIO.LOW)
        else:
            # Turn LED on
            GPIO.output(LED_PIN, GPIO.HIGH)

        # Short delay to debounce button press
        time.sleep(0.01)

except KeyboardInterrupt:
    # Clean up GPIO on Ctrl+C exit
    GPIO.cleanup()
```

Explanation of the Inverted Logic

- **if button_state == GPIO.LOW:**

- When the button is pressed, button_state will be GPIO.LOW. According to the new logic, the LED should be **off** in this case, so we use GPIO.output(LED_PIN, GPIO.LOW).

- **else:**
  - When the button is not pressed, button_state will be GPIO.HIGH. According to the new logic, the LED should be **on** in this case, so we use GPIO.output(LED_PIN, GPIO.HIGH).

Summary

- **Button Pressed**: LED turns **off**.
- **Button Not Pressed**: LED turns **on**.

This modified code will ensure that the LED turns off when the pushbutton is pressed and turns on when the button is not pressed.

To achieve the behavior where the LED turns **on** when the pushbutton is pressed and **off** when the button is not pressed, you need to set the logic correctly in your Python script for the Raspberry Pi.

```
import RPi.GPIO as GPIO
import time

# GPIO pin numbers
BUTTON_PIN = 17
LED_PIN = 27

# Set up GPIO mode
GPIO.setmode(GPIO.BCM)  # Use Broadcom pin numbering
GPIO.setup(LED_PIN, GPIO.OUT)  # Set LED pin as an output
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set button pin as an input with internal pull-up resistor

try:
    while True:
        # Read the state of the pushbutton
        button_state = GPIO.input(BUTTON_PIN)

        # Check if the pushbutton is pressed (LOW when pressed)
        if button_state == GPIO.LOW:
```

```
        # Turn LED on
        GPIO.output(LED_PIN, GPIO.HIGH)
    else:
        # Turn LED off
        GPIO.output(LED_PIN, GPIO.LOW)

    # Short delay to debounce button press
    time.sleep(0.01)

except KeyboardInterrupt:
    # Clean up GPIO on Ctrl+C exit
    GPIO.cleanup()
```

Explanation

1. **GPIO.setup(LED_PIN, GPIO.OUT)**: Configures the pin connected to the LED as an output.
2. **GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)**: Configures the pin connected to the pushbutton as an input with an internal pull-up resistor. This means the pin reads GPIO.HIGH when the button is not pressed and GPIO.LOW when it is pressed.
3. **button_state = GPIO.input(BUTTON_PIN)**: Reads the current state of the pushbutton.
4. **if button_state == GPIO.LOW:**
    - If the button is pressed (GPIO.LOW), the LED is turned on with GPIO.output(LED_PIN, GPIO.HIGH).
5. **else:**
    - If the button is not pressed (GPIO.HIGH), the LED is turned off with GPIO.output(LED_PIN, GPIO.LOW).
6. **time.sleep(0.01)**: Adds a short delay to debounce the button press, avoiding multiple triggers from a single press.
7. **GPIO.cleanup()**: Ensures that the GPIO pins are cleaned up properly when the script is interrupted (e.g., with Ctrl+C).

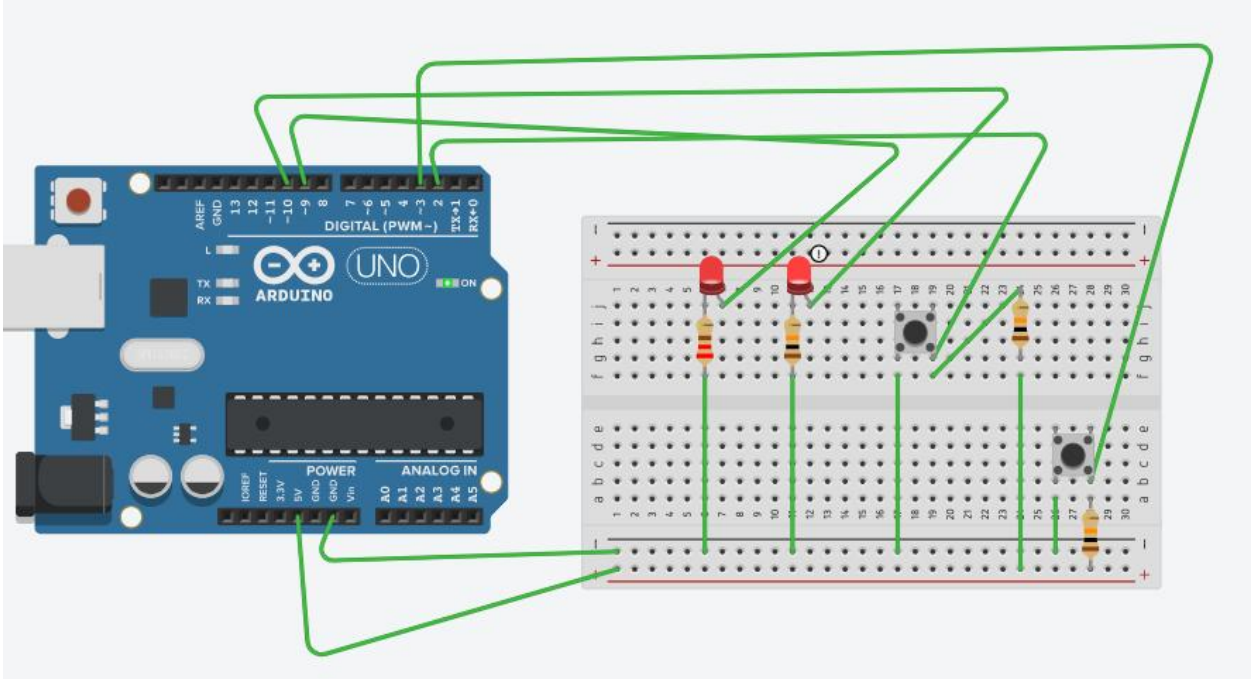
Summary

- **Button Pressed**: LED is **on**.
- **Button Not Pressed**: LED is **off**.

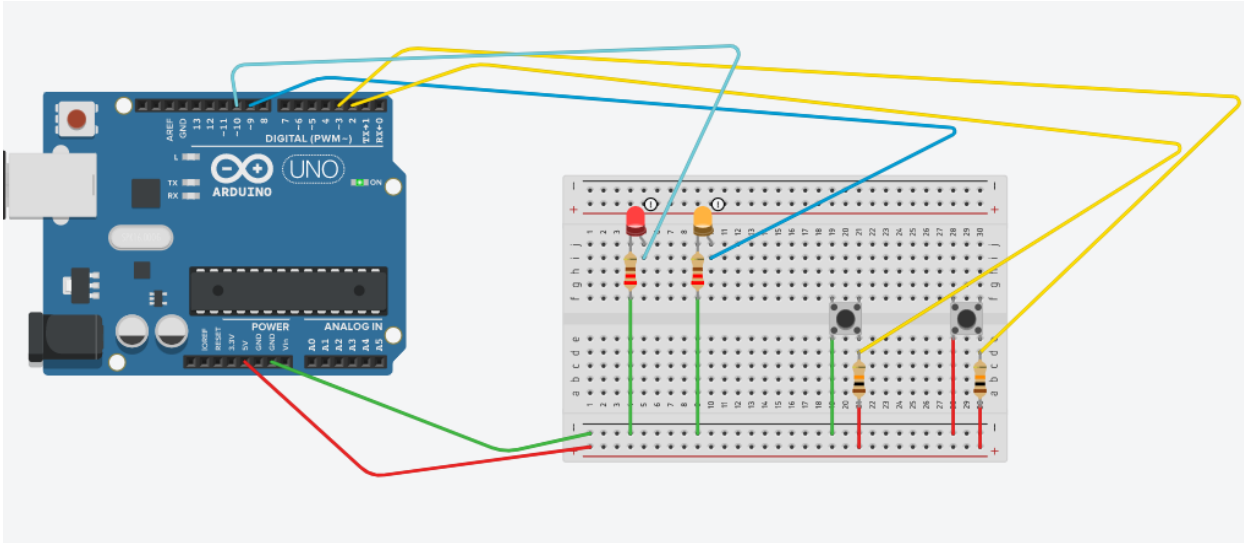
program to demonstrate two leds with two buttons using **Arduino Uno**

Off LED when pushbutton pressed down

The pushbutton is not arranged properly, but it is still working because all its pins are in the same position, connecting the two terminals in the same line. its arrangement is not ideal because it connects terminals inappropriately.



The proper way to arrange the pushbutton is to place it across the middle gap of the breadboard or into the Dual In-line Package (DIP) socket. This ensures that the pins do not connect to the same terminal.



1. Set Up Your Circuit in Tinkercad

1. **Create a New Circuit:**
   - Go to Tinkercad, sign in, and click on “Circuits” in the left menu.
   - Click the “Create new Circuit” button.
2. **Add Components:**
   - **Arduino Uno**: Search for "Arduino Uno" in the components menu and drag it onto the workspace.
   - **2 LEDs**: Search for "LED" and drag two LEDs onto the workspace.
   - **2 Pushbuttons**: Search for "Pushbutton" and drag two pushbuttons onto the workspace.
   - **4 Resistors**: Search for "Resistor" and add four resistors (220Ω for LEDs, 10kΩ for pull-down resistors).
   - **Breadboard**: Drag a breadboard onto the workspace for organizing your components.
   - **Jumper Wires**: Use jumper wires to make connections.
3. **Wire the LEDs:**
   - Place each LED on the breadboard.
   - Connect the positive (long) leg of each LED to a digital pin on the Arduino (e.g., pin 9 for LED1 and pin 10 for LED2) through a 220Ω resistor.
   - Connect the negative (short) leg of each LED to the ground (GND) on the Arduino.
4. **Wire the Pushbuttons:**
   - Place each pushbutton on the breadboard.
   - Connect one side of each pushbutton to the ground (GND) on the Arduino.
   - Connect the other side of each pushbutton to separate digital pins on the Arduino (e.g., pin 2 for Button1 and pin 3 for Button2).
   - Add a 10kΩ resistor between the GPIO pin and ground for each button to act as a pull-down resistor. This ensures a stable LOW state when the button is not pressed.
5. **Check Connections:**

- Ensure all connections are correct and secure according to the wiring described.

2. Write the Code

1. **Open the Code Editor:**
   - Click on the "Code" button in the upper right corner of the Tinkercad interface.
   - Choose "Blocks + Text" mode to see both block-based and text-based code.
2. **Write the Arduino Code:**

```
const int button1Pin = 2;    // the number of the first pushbutton pin
const int button2Pin = 3;    // the number of the second pushbutton pin
const int led1Pin = 9;       // the number of the first LED pin
const int led2Pin = 10;      // the number of the second LED pin

void setup() {
  // Initialize the LED pins as outputs
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  // Initialize the pushbutton pins as inputs
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);
}

void loop() {
  // Read the state of each pushbutton
  int button1State = digitalRead(button1Pin);
  int button2State = digitalRead(button2Pin);

  // Control the LEDs based on the pushbutton states
  if (button1State == HIGH) {
    digitalWrite(led1Pin, HIGH);  // Turn LED1 on
  } else {
```

```
  digitalWrite(led1Pin, LOW);   // Turn LED1 off
 }

 if (button2State == HIGH) {
  digitalWrite(led2Pin, HIGH);  // Turn LED2 on
 } else {
  digitalWrite(led2Pin, LOW);   // Turn LED2 off
 }

 // Add a short delay to make button presses more stable
 delay(50);
}
```

Explanation of the Code

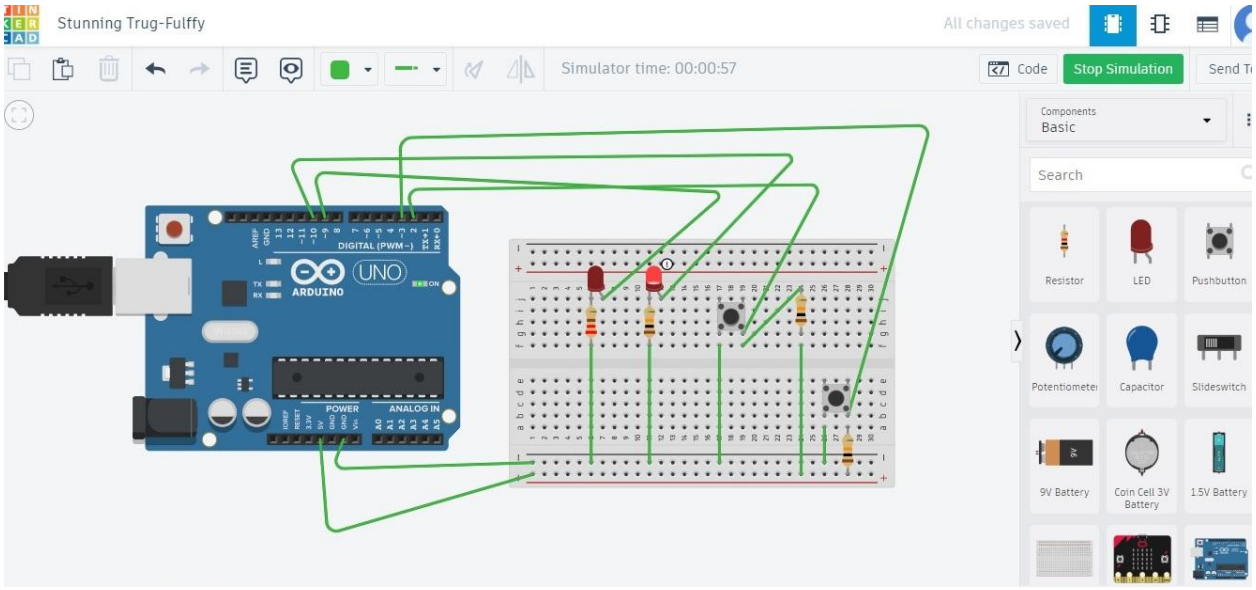
1. **Variables:**
    - button1Pin, button2Pin: Pin numbers for the pushbuttons.
    - led1Pin, led2Pin: Pin numbers for the LEDs.
2. **setup() Function:**
    - Configures LED pins as outputs and button pins as inputs.
3. **loop() Function:**
    - Reads the state of each button.
    - Turns each LED on or off based on the corresponding button state.
    - Adds a small delay to improve button press stability.

Summary

- **Button1** controls **LED1**: When Button1 is pressed, LED1 turns on.
- **Button2** controls **LED2**: When Button2 is pressed, LED2 turns on

ON LED when pushbutton pressed down

To modify the code so that the LEDs turn **on** when the pushbuttons are pressed **down** and turn **off** when the pushbuttons are not pressed, you should adjust the condition in the if statements. In the original code, HIGH means the button is not pressed, while LOW indicates it is pressed. You want the LEDs to light up when the buttons are pressed, so you'll need to invert the logic.




```
const int button1Pin = 2;    // the number of the first pushbutton pin
const int button2Pin = 3;    // the number of the second pushbutton pin
const int led1Pin = 9;       // the number of the first LED pin
const int led2Pin = 10;      // the number of the second LED pin

void setup() {
  // Initialize the LED pins as outputs
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  // Initialize the pushbutton pins as inputs
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);
}

void loop() {
  // Read the state of each pushbutton
  int button1State = digitalRead(button1Pin);
  int button2State = digitalRead(button2Pin);
```

```
  // Control the LEDs based on the pushbutton states
  if (button1State == LOW) {   // When Button1 is pressed (LOW), turn LED1 on
    digitalWrite(led1Pin, HIGH);
  } else {                     // When Button1 is not pressed (HIGH), turn LED1 off
    digitalWrite(led1Pin, LOW);
  }

  if (button2State == LOW) {   // When Button2 is pressed (LOW), turn LED2 on
    digitalWrite(led2Pin, HIGH);
  } else {                     // When Button2 is not pressed (HIGH), turn LED2 off
    digitalWrite(led2Pin, LOW);
  }

  // Add a short delay to make button presses more stable
  delay(50);
}
```

Explanation of Changes

1. **button1State == LOW and button2State == LOW:**
    - This condition checks if the button is pressed. When the button is pressed, the buttonState reads LOW due to the pull-up resistor.
2. **LED Control:**
    - **When button1State == LOW**: The LED connected to led1Pin will be turned on (HIGH).
    - **When button1State == HIGH**: The LED connected to led1Pin will be turned off (LOW).
    - **When button2State == LOW**: The LED connected to led2Pin will be turned on (HIGH).
    - **When button2State == HIGH**: The LED connected to led2Pin will be turned off (LOW).

Summary

With this updated code:

- **Button1 Pressed**: LED1 turns **on**.

- **Button1 Not Pressed**: LED1 turns **off**.
- **Button2 Pressed**: LED2 turns **on**.
- **Button2 Not Pressed**: LED2 turns **off**.

program to demonstrate two leds with two buttons using Raspberry Pi - ON LED when pushbutton pressed down

Python code for a Raspberry Pi so that two LEDs turn **on** when their respective pushbuttons are pressed and turn **off** when the buttons are not pressed, you'll need to adjust the logic accordingly. The pushbutton's state will be read as LOW when pressed (due to the internal pull-up resistor configuration) and HIGH when not pressed.

```
import RPi.GPIO as GPIO
import time

# GPIO pin numbers
BUTTON1_PIN = 17
BUTTON2_PIN = 27
LED1_PIN = 22
LED2_PIN = 23

# Set up GPIO mode
GPIO.setmode(GPIO.BCM)  # Use Broadcom pin numbering
GPIO.setup(LED1_PIN, GPIO.OUT)  # Set LED1 pin as an output
GPIO.setup(LED2_PIN, GPIO.OUT)  # Set LED2 pin as an output
GPIO.setup(BUTTON1_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set button1 pin as an input with internal pull-up resistor
GPIO.setup(BUTTON2_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set button2 pin as an input with internal pull-up resistor

try:
    while True:
```

```
        # Read the state of each pushbutton
        button1_state = GPIO.input(BUTTON1_PIN)
        button2_state = GPIO.input(BUTTON2_PIN)

        # Control the LEDs based on the pushbutton states
        if button1_state == GPIO.LOW:  # When Button1 is pressed (LOW), turn LED1 on
            GPIO.output(LED1_PIN, GPIO.HIGH)
        else:  # When Button1 is not pressed (HIGH), turn LED1 off
            GPIO.output(LED1_PIN, GPIO.LOW)

        if button2_state == GPIO.LOW:  # When Button2 is pressed (LOW), turn LED2 on
            GPIO.output(LED2_PIN, GPIO.HIGH)
        else:  # When Button2 is not pressed (HIGH), turn LED2 off
            GPIO.output(LED2_PIN, GPIO.LOW)

        # Short delay to debounce button press
        time.sleep(0.01)

except KeyboardInterrupt:
    # Clean up GPIO on Ctrl+C exit
    GPIO.cleanup()
```

Explanation of the Code

1. **Import Libraries:**
    - RPi.GPIO for GPIO control.
    - time for adding delays.
2. **Define Pin Numbers:**
    - BUTTON1_PIN and BUTTON2_PIN: GPIO pins connected to the pushbuttons.
    - LED1_PIN and LED2_PIN: GPIO pins connected to the LEDs.
3. **Set Up GPIO:**

- GPIO.setmode(GPIO.BCM): Use Broadcom pin numbering.
- GPIO.setup(LED1_PIN, GPIO.OUT): Set the LED pins as outputs.
- GPIO.setup(BUTTON1_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP): Set the button pins as inputs with internal pull-up resistors, meaning the pin reads HIGH when the button is not pressed and LOW when it is pressed.

4. **Read Button States:**
   - button1_state = GPIO.input(BUTTON1_PIN): Reads the state of Button1.
   - button2_state = GPIO.input(BUTTON2_PIN): Reads the state of Button2.
5. **Control LEDs:**
   - If button1_state is GPIO.LOW, turn LED1 on (GPIO.HIGH). If button1_state is GPIO.HIGH, turn LED1 off (GPIO.LOW).
   - If button2_state is GPIO.LOW, turn LED2 on (GPIO.HIGH). If button2_state is GPIO.HIGH, turn LED2 off (GPIO.LOW).
6. **Debounce Delay:**
   - time.sleep(0.01): Adds a short delay to debounce the button press and prevent multiple triggers from a single press.
7. **Exception Handling:**
   - GPIO.cleanup(): Cleans up GPIO settings on script interruption (Ctrl+C).

Summary

- **Button1 Pressed**: LED1 turns **on**.
- **Button1 Not Pressed**: LED1 turns **off**.
- **Button2 Pressed**: LED2 turns **on**.
- **Button2 Not Pressed**: LED2 turns **off**.

This modified code ensures that each LED lights up when its corresponding button is pressed and turns off when the button is released.

program to demonstrate two leds with two buttons using Raspberry Pi - OFF LED when pushbutton pressed down

```
import RPi.GPIO as GPIO
import time

# GPIO pin numbers
BUTTON1_PIN = 17  # GPIO pin for the first pushbutton
BUTTON2_PIN = 27  # GPIO pin for the second pushbutton
LED1_PIN = 22     # GPIO pin for the first LED
LED2_PIN = 23     # GPIO pin for the second LED

# Set up GPIO mode
GPIO.setmode(GPIO.BCM)  # Use Broadcom (BCM) pin numbering

# Set up pins
GPIO.setup(LED1_PIN, GPIO.OUT)  # Set LED1 pin as an output
GPIO.setup(LED2_PIN, GPIO.OUT)  # Set LED2 pin as an output
GPIO.setup(BUTTON1_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set button1 pin as an input with pull-up resistor
GPIO.setup(BUTTON2_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set button2 pin as an input with pull-up resistor

try:
    while True:
        # Read the state of each pushbutton
        button1_state = GPIO.input(BUTTON1_PIN)
        button2_state = GPIO.input(BUTTON2_PIN)

        # Control the LEDs based on the pushbutton states
        if button1_state == GPIO.LOW:  # Button1 pressed (LOW)
```

```
        GPIO.output(LED1_PIN, GPIO.HIGH)  # Turn LED1 on
    else:
        GPIO.output(LED1_PIN, GPIO.LOW)   # Turn LED1 off

    if button2_state == GPIO.LOW:  # Button2 pressed (LOW)
        GPIO.output(LED2_PIN, GPIO.HIGH)  # Turn LED2 on
    else:
        GPIO.output(LED2_PIN, GPIO.LOW)   # Turn LED2 off

    # Add a short delay to debounce button presses
    time.sleep(0.05)

except KeyboardInterrupt:
    # Clean up GPIO on Ctrl+C exit
    GPIO.cleanup()
```

Explanation

1. **Import Libraries:**
    - RPi.GPIO for controlling the GPIO pins.
    - time for delays.
2. **Define Pin Numbers:**
    - Use the GPIO pin numbers (BUTTON1_PIN, BUTTON2_PIN, LED1_PIN, LED2_PIN) according to the Raspberry Pi GPIO pinout.
3. **Set Up GPIO Pins:**
    - GPIO.setmode(GPIO.BCM): Use Broadcom pin numbering.
    - GPIO.setup(LED1_PIN, GPIO.OUT): Configure the LED pins as outputs.
    - GPIO.setup(BUTTON1_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP): Configure the button pins as inputs with internal pull-up resistors.
4. **Read Button States:**
    - GPIO.input(BUTTON1_PIN): Reads the state of the pushbutton.
    - GPIO.input(BUTTON2_PIN): Reads the state of the second pushbutton.

5. **Control LEDs:**
    - If the button state is GPIO.LOW, turn the corresponding LED on (GPIO.HIGH).
    - If the button state is GPIO.HIGH, turn the LED off (GPIO.LOW).
6. **Debounce Delay:**
    - time.sleep(0.05): A short delay to debounce the button press, making it more stable.
7. **Exception Handling:**
    - GPIO.cleanup(): Cleans up GPIO settings on script interruption.