1. Room Temperature Measurement

Objective: Measure room temperature using a TMP36 sensor and display it on the serial console.

Steps:

1. **Place Components**: Add an Arduino, TMP36 sensor, and connect the components in Tinkercad.
2. **Connect Sensor Power**: Attach the TMP36 sensor's VCC pin to the Arduino's `5V` pin and GND to `GND`.
3. **Connect Analog Pin**: Connect the TMP36's output pin to the Arduino's analog input `A0`.
4. **Initialize Serial Monitor**: In the code, add `Serial.begin(9600);` in the `setup()` function to initialize serial communication.
5. **Read Sensor Value**: In the `loop()` function, use `analogRead(A0)` to read the TMP36 sensor value.
6. **Convert to Celsius**: Convert the analog reading to temperature in Celsius using `tempC = (voltage - 0.5) * 100;`.
7. **Display on Serial Console**: Print the temperature to the serial monitor with `Serial.println(tempC);`.
8. **Add Delay**: Add a delay of 1 second using `delay(1000);` for continuous readings.
9. **Run Simulation**: Start the simulation in Tinkercad and open the serial monitor to view the temperature readings.
10. **Observe Output**: Verify that the temperature data updates every second.

---

2. Gas Detection

Objective: Detect the presence of gas (like smoke) using an MQ sensor and display the status on the serial console.

Steps:

1. **Place Components**: Add an Arduino, MQ2 gas sensor, and LED to the workspace.
2. **Connect Power**: Connect the MQ2 sensor's VCC to `5V` and GND to `GND` on the Arduino.
3. **Connect Analog Pin**: Connect the MQ2 sensor's analog output pin to the Arduino's `A0`.
4. **Initialize Serial Monitor**: Use `Serial.begin(9600);` in the `setup()` function.
5. **Read Sensor Value**: In the `loop()` function, use `analogRead(A0)` to read the sensor value.
6. **Set Threshold**: Define a gas threshold value (e.g., 300) to detect gas presence.
7. **Check Gas Levels**: Use an `if` condition to compare the sensor value with the threshold.

8. **Display Result**: If gas is detected, print `"Gas Detected!"`; otherwise, print `"No Gas"` to the serial monitor.
9. **Add Delay**: Include `delay(1000);` to update every second.
10. **Run and Observe**: Start the simulation and monitor the serial console to see gas detection status.

---

3. Obstacle Detection with Infrared Sensors

Objective: Detect obstacles using an IR sensor and display the approximate distance on the serial console.

Steps:

1. **Place Components**: Add an Arduino and an IR sensor to Tinkercad.
2. **Connect Power**: Attach the IR sensor's VCC to `5V` and GND to `GND` on the Arduino.
3. **Connect Digital Pin**: Connect the IR sensor's output pin to Arduino's digital pin, e.g., `D2`.
4. **Initialize Serial Monitor**: In `setup()`, add `Serial.begin(9600);`.
5. **Read IR Sensor State**: In the `loop()`, use `digitalRead(D2)` to check if an obstacle is detected.
6. **Define Threshold**: Determine a threshold for distance (e.g., if signal is HIGH, obstacle is close).
7. **Display Detection Status**: Use `Serial.println("Obstacle Detected")` if an obstacle is detected.
8. **No Obstacle Condition**: Print `"No Obstacle"` if no obstacle is detected.
9. **Add Delay**: Use `delay(500);` to check every half second.
10. **Run Simulation**: Start simulation and view the serial monitor for obstacle detection feedback.

Using an **ultrasonic distance sensor** for the fourth experiment in Tinkercad allows you to measure the distance of an object more accurately than using a potentiometer. Here's a revised 10-step procedure to implement this experiment with an ultrasonic sensor, such as the **HC-SR04**.

---

4. Distance Measurement with Ultrasonic Sensor

Objective: Measure the distance of an object using an ultrasonic sensor and display the result on the serial console to simulate speed or distance changes.

Steps:

1. **Place Components**: Add an Arduino and an HC-SR04 ultrasonic sensor in Tinkercad.
2. **Connect Sensor Power**: Connect the **VCC** pin of the ultrasonic sensor to the Arduino's `5V` and **GND** to `GND` .
3. **Connect Trigger and Echo Pins**:
   - Connect the **Trig** pin of the sensor to a digital pin on the Arduino, e.g., `D9` .
   - Connect the **Echo** pin to another digital pin, e.g., `D10` .
4. **Initialize Serial Monitor**: In the `setup()` function, add `Serial.begin(9600);` to enable serial communication.
5. **Configure Sensor Pins**: In `setup()` , set the **Trig** pin as `OUTPUT` and the **Echo** pin as `INPUT` :
6. **Send Trigger Pulse**: In `loop()` , send a 10-microsecond pulse to the Trig pin to start the measurement:
7. **Read Echo Pulse**: Use `pulseIn(10, HIGH);` to measure the duration it takes for the sound wave to return after hitting an object:
8. **Calculate Distance**: Convert the pulse duration to a distance in centimeters:
9. **Display Distance**: Print the distance to the serial monitor with `Serial.println(distance);` .
   - You can use this as an approximation of "speed" by noting how distance changes over time.
10. **Add Delay**: Use `delay(500);` to update every half second and repeat the measurement process.

---

5. Smart Home System using Wifi Module

Procedure

1. **Prepare Components**: In Tinkercad, add an Arduino and an LED. Connect the LED's positive (anode) leg to a digital pin, such as pin 13, and the negative (cathode) leg to the ground (GND) via a resistor.

2. **Define the LED Pin**: In the Arduino sketch, specify the digital pin (e.g., pin 13) where the LED is connected. Set this pin as the LED control pin.

3. **Set Up Serial Communication**: In the `setup()` function, initialize serial communication with the `Serial.begin(9600);` command. This allows the Arduino to receive commands from the serial monitor.

4. **Configure LED Pin as Output**: Also in the `setup()` function, set the LED pin as an output using `pinMode()`. This enables the Arduino to control the LED's state (on or off).

5. **Check for Serial Input**: In the `loop()` function, use `Serial.available()` to check if any data has been entered in the serial monitor. This tells the Arduino to listen for commands.

6. **Read the Command**: When a command is detected, use `Serial.readString()` to read the input from the serial monitor. This command reads all characters entered until a newline is detected.

7. **Interpret Command to Turn LED On**: If the received command is `"s"`, turn on the LED by setting the LED pin to HIGH. This simulates turning on a light based on a user instruction.

8. **Interpret Command to Turn LED Off**: If the command received is `"e"`, turn off the LED by setting the LED pin to LOW. This simulates turning off the light on user command.

9. **Test the Setup**: Start the simulation and open the serial monitor. Type `"s"` to turn the LED on, and type `"e"` to turn it off, then press Enter.

10. **Observe the LED Behavior**: Verify that the LED responds accurately to each command. Use different commands to ensure the LED only responds to `"s"` (on) and `"e"` (off).

---

6. PubNub Dashboard for Student Details

Objective: Send student data (name, age, grade) to a PubNub server and retrieve it back for display on a common dashboard.

Procedure

1. **Create a PubNub Account**: Go to PubNub's website and sign up or log in to your account. Once logged in, create a new PubNub app for this project.

2. **Obtain PubNub Keys**: In the app dashboard, find and copy your **Publish Key** and **Subscribe Key**. These will be needed for your HTML code to send and retrieve data.

3. **Setup HTML Page**: Create a simple HTML file named `dashboard.html`. Add input fields for **Name**, **Age**, and **Grade**, and a button to submit data.

4. **Include PubNub JavaScript SDK**: Add a `<script>` tag to include the PubNub SDK in your HTML. This allows your webpage to interact with the PubNub server.

5. **Initialize PubNub in JavaScript**: In your HTML file, add a `<script>` section to initialize PubNub with your publish and subscribe keys.

6. **Send Data to PubNub**:

   - Add an event listener to your "Submit" button. When clicked, it captures the name, age, and grade input values.
   - Publish this data to a PubNub channel (e.g., `"student_channel"`).

7. **Subscribe to the PubNub Channel**:

   - Use PubNub's `subscribe` function to listen for new messages on the `"student_channel"`.
   - Every time data is published to this channel, it will be received and displayed on the HTML page.

8. **Display Received Data**:

   - When a message is received, update a section of the HTML page (e.g., a `<div>` or `<table>`) to show the latest student details.

9. **Test the Setup**: Open the `dashboard.html` file in a browser. Enter sample data (e.g., "Alice, 12, 7th Grade") and submit. The data should appear immediately in the display section of the dashboard.

10. **Observe Real-Time Updates**: Add more entries or open multiple instances of `dashboard.html` in different browser tabs. All connected clients should see new student data in real time.