Introduction to Programming
Lab: Understanding Loops

Vikas Thammanna Gowda

07/29/2025

Name: _____

Introduction

In this lab, you will build and program a Raspberry Pi circuit to create two lighting projects: a circular LED sequence and a traffic light simulator with pedestrian crossing.

- **Part 1:** You will be provided with the *complete code, circuit diagram, setup, and wiring instructions*, along with a *brief in-class demonstration*. In this part, four LEDs will blink in a loop, one after the other, creating the illusion of a moving light.
- **Part 2:** A collaborative activity, where you will work in groups to reconfigure the circuit to simulate the behavior of a basic traffic light with pedestrian crossing. You will modify the code and test your design as a team, applying your understanding of loops and GPIO control.

Learning Objectives

- Understand the purpose and behavior of `while` loops.
- Apply `while (true)` loops to create continuous, repeating LED patterns.
- Observe how loop structure controls the timing and order of physical outputs.
- Modify loop contents to change the LED blinking sequence and timing.
- Use `sleep()` delays within loops to control the pace of execution.
- Recognize how loops interact with real-world hardware through GPIO control.
- Implement `try` and `except` blocks to safely exit infinite loops.

Required Components:

- Raspberry Pi (any model with 40 GPIO pins) with Raspbian/Raspberry Pi OS installed.
- Breadboard and jumper wires.
- 4 LEDs for Part 1. For Part 2, you'll specifically use 3 LEDs (one red, one yellow, one green to mimic a traffic light).
- 4 Resistors (220 Ω or 330 Ω are typical) - one for each LED to limit current.
- 1 Button

Part 1: Four LEDs in a Circular Sequence

In this part, you will build a circuit with four LEDs and run the program to turn them on and off in a repeating loop. The LEDs will light up one by one in order, creating a chaser or "spinning" effect (imagine a single light that appears to move from one LED to the next in a circle). This will involve a continuous loop (to keep the lights blinking until you stop the program) and timing delays to control the speed of the chase.

Illustration

Follow these steps to assemble the LED circuit. **Make sure your Raspberry Pi is shut down or powered off while wiring the circuit to avoid any accidental short circuits or damage.**

1. **Place the LEDs on the breadboard:** Arrange the four LEDs in a roughly circular or square layout on your breadboard (this is just for visual effect, the exact placement isn't critical as long as each LED is in a separate row). Ensure the legs of each LED are in different rows so they aren't accidentally connected. Recall that an LED has polarity: the longer leg is the positive anode, and the shorter leg is the negative cathode. The anode will connect to a GPIO pin (through a resistor), and the cathode will connect to the Raspberry Pi's ground. (Tip: If you forget which leg is which, note that the flat side of the LED casing corresponds to the cathode, and the cathode leg is usually shorter.)

2. **Add resistors in series with each LED:** Connect one end of a resistor to the anode of each LED. The resistor can go in the same row as the LED's anode lead. The other end of the resistor will later go to a GPIO pin via a jumper wire. Using 220 or 330 Ω resistors for standard 5mm LEDs will limit the current and protect the LED and GPIO pin. All four LEDs need their own resistor. (It doesn't matter which end of the LED the resistor is on, as long as it's in series - either between GPIO and anode, or between cathode and ground - the effect is the same.)

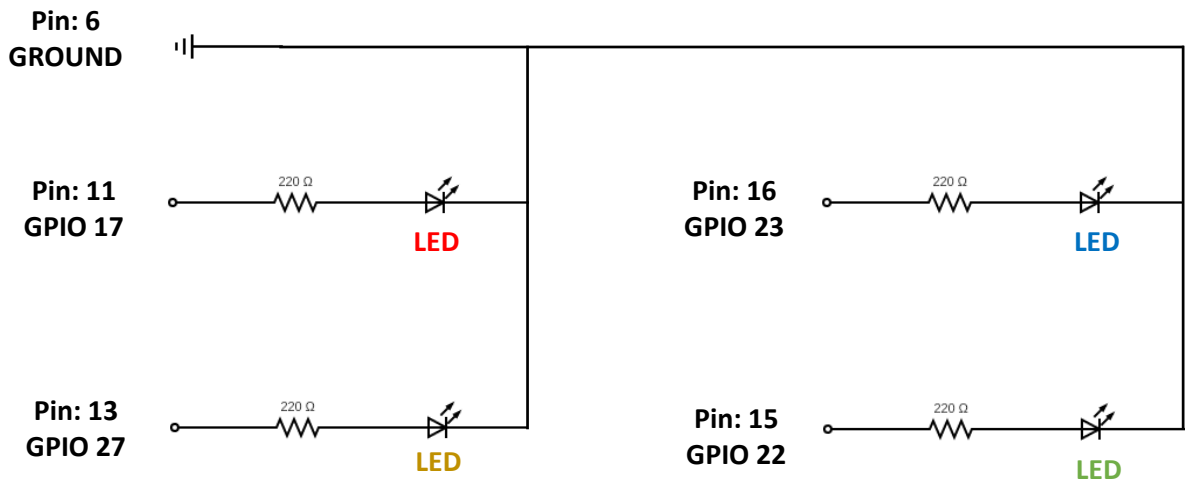

Figure 1: Circuit Diagram.

3. **Connect LED cathodes to ground:** Connect the negative leg (cathode) of each LED to the Pi's ground. An easy way to do this is to use a ground rail on the breadboard. First, use a jumper wire from one of the Pi's GND pins (there are several GND pins; one convenient GND is physical pin 6 on the GPIO header) to a hole in the breadboard and designate that entire row as "ground". Then connect a short jumper from each LED's cathode row to that ground rail. Now

all LED cathodes are tied to the Pi's ground (they can share the same ground rail since ground is common).

4. **Connect LED anodes to GPIO pins:** Now connect the free end of each resistor (the end not connected to an LED yet) to the appropriate Raspberry Pi GPIO output pin using jumper wires. We will use the Broadcom (BCM) numbering for GPIO pins (this is the default numbering system in the gpiozero library). The four GPIO pins we'll use are:

   - GPIO17 - physical pin 11 on the Pi's header
   - GPIO27 - physical pin 13
   - GPIO22 - physical pin 15
   - GPIO23 - physical pin 16

   Each resistor end gets one jumper wire to one of these GPIO pins. For example, connect the resistor from LED1's anode to GPIO17, LED2's resistor to GPIO27, LED3's resistor to GPIO22, and LED4's resistor to GPIO23. These four GPIO pins are all on the same side of the header (they are convenient because they are adjacent pins 11, 13, 15, 16). Double-check your connections: each LED's path should be: GPIO pin ¿ resistor -¿ LED anode, then LED cathode -¿ ground. If you wire it this way, setting a GPIO pin "HIGH" (on) will forward-bias the LED (current flows from the 3.3V output pin through the LED to ground) and the LED will light. Setting the pin "LOW" (off) will stop current flow, turning the LED off.

5. **Final check:** Ensure no two LED leads are accidentally in the same row (which would short them), and verify each LED has one side to ground and one side to a unique GPIO pin (through a resistor). Also ensure your Pi's ground pin is connected to the LED cathodes. The wiring should resemble four "spokes" from the Pi to each LED, with a common ground. Once everything looks correct, you can power on your Raspberry Pi for the next step.

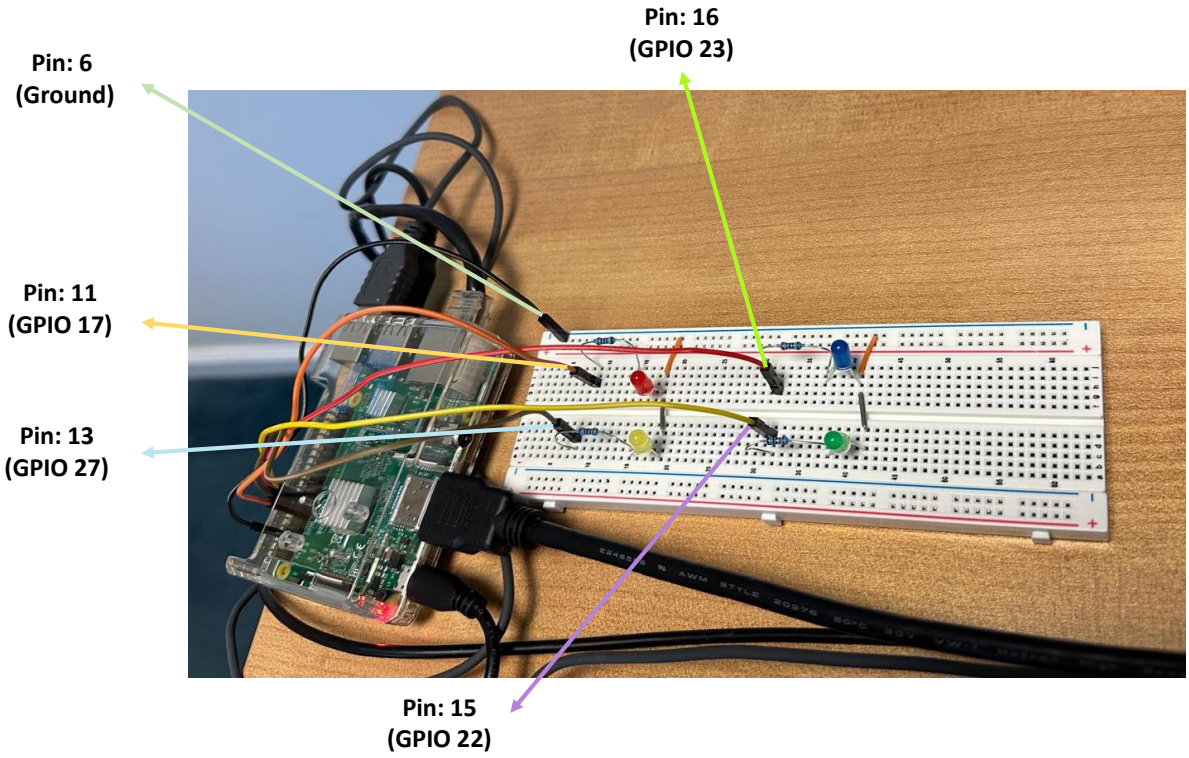

Figure 2: Wiring set-up.

Run and Observe

It's time to test the circuit and code. Make sure your Raspberry Pi is powered on and the circuit is connected as described. Run the privided code.

- LEDs should light one after another in a circular pattern.
- Press **Ctrl+C** to stop and see "Cleaned up" in the terminal.

Part 2: Traffic Light Simulation

Now that you have mastered basic LED control and looping, let's apply those skills to a real-world scenario: a traffic light system. Real traffic lights cycle through green, yellow (amber), and red lights with specific timing. In this part, you will modify your hardware setup to use three LEDs—typically one of each color: green, yellow, and red—and update your code to simulate a standard traffic signal pattern. This exercise demonstrates how the same circuit and code structure can be reused and adapted for different purposes simply by changing the sequence and timing logic.

Now, consider how you might enhance this setup by adding a pedestrian crossing button. For instance, when the button is pressed, the traffic light should turn red (if it isn't already), allowing pedestrians to cross safely.

Circuit Diagram

Give the circuit diagram:

Reflection and Analysis

1. In your own words, what does while true: do in the program, and why was it used in both parts of this lab? What would happen if we used a for loop with a fixed range instead?

2. If you wanted the LED chase in Part 1 to run twice as fast, what would you change in the code? Likewise, how would you modify the traffic light timing to make the green light last 10 seconds instead of 5? (Describe which variables or values to change.)

3. Our programs turn off all the LEDs in the finally block when exiting. Why is this important? What might you observe on the hardware if the program was stopped without turning off the LEDs?

4. **Further extension:** Discuss how this project can be extended.