Introduction to Programming
Lab: Understanding Functions

Vikas Thammanna Gowda

07/29/2025

Name: _____

Introduction

In this lab, you will build and program a Raspberry Pi circuit to play a Rock-Paper-Scissors game against the computer using three buttons, LEDs, and a character LCD.

- **Part 1:** You will be provided with the *complete code, circuit diagram, setup, and wiring instructions*, along with a *brief in-class demonstration*. You will press a button to make a move; the LCD will display your choice, the computer's choice, and the result.
- **Part 2:** Enhance the code by refactoring repeated logic into functions and adding multiple game modes (Sudden Death, Best of 3, First to 3 Wins). This activity illustrates how user-defined functions promote reuse and clarity.

Learning Objectives

- Understand definition and invocation of Python functions.
- Apply functions to encapsulate game logic and avoid repetition.
- Observe how loop structure controls the timing and order of physical outputs.
- Integrate user input (buttons) and output (LCD, LEDs) in an event-driven program
- Design program flow for different game-mode rules.
- Learn to use the `finally` block to ensure cleanup actions (like turning off LEDs) are always executed.

Required Components:

- Raspberry Pi (any model with 40 GPIO pins) with Raspbian/Raspberry Pi OS installed.
- Breadboard and jumper wires.
- 3 bottons, LCD display for Part 1.
- For Part 2, you will choose the componenets to enhance your project, LEDs, resistors, and additional buttons as needed.

Part 1: Single Round RPS

In Part 1, students use three GPIO-connected push-buttons and a 20×4 character LCD to implement a single-round Rock-Paper-Scissors game entirely in software. As soon as the script starts, it blocks waiting for one of the buttons to be pressed—each button representing rock, paper, or scissors—and then immediately sends that choice to the LCD. The program simultaneously generates a random "computer" choice, and the LCD briefly displays both selections (for example, "You: Paper" on the top line and "CPU: Scissors" on the bottom line). After a one-second pause, the script computes the outcome using a simple modulo-3 check and replaces the display with "You Win," "You Lose," or "Draw" for two seconds before clearing the screen and returning to wait for the next button press. This exercise gives students hands-on practice with reading button input, writing text to an LCD, blocking for user input, and implementing core game-logic in Python.

Illustration

Follow these steps to assemble the circuit. These instructions describe how to wire three tactile push-buttons to a Raspberry Pi using its GPIO pins and the breadboard's ground rail. Each button will serve as an input for the Rock-Paper-Scissors game: Button 1 for Rock, Button 2 for Paper, and Button 3 for Scissors. Internal pull-up resistors on the Pi will hold each input high until the button is pressed, pulling it to ground. **Make sure your Raspberry Pi is shut down or powered off while wiring the circuit to avoid any accidental short circuits or damage.**

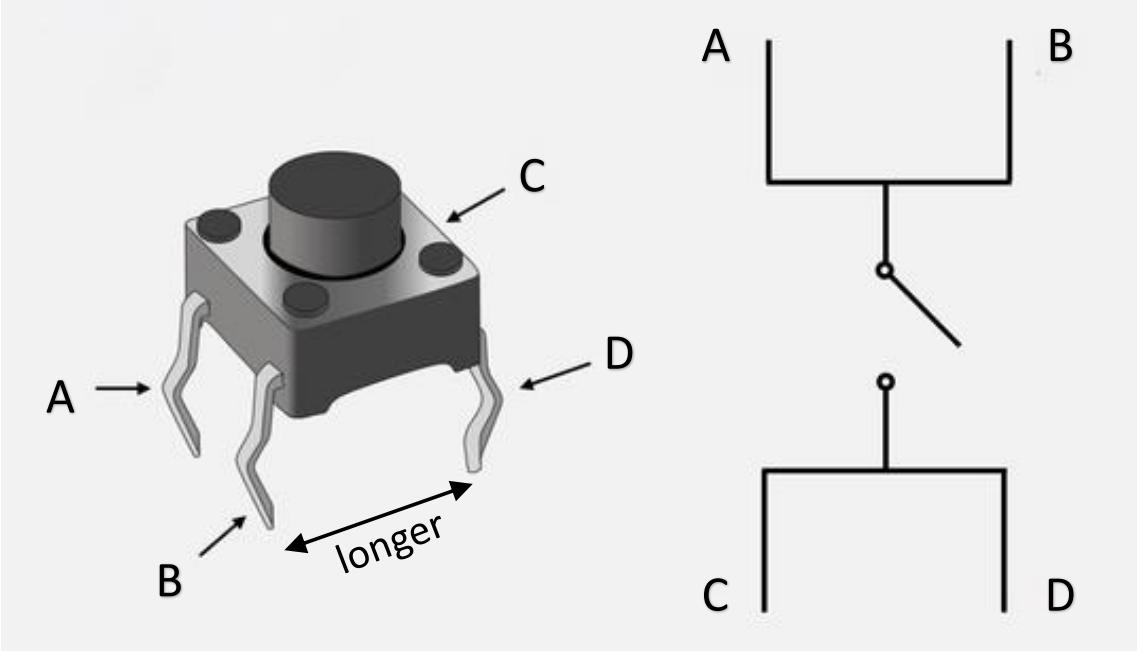

Figure 1: Button internal circuit.

Button Wiring

1. **Prepare the Breadboard:** Before wiring any components, establish a common ground reference between the Raspberry Pi and the breadboard. This ensures that when a button is pressed, the Pi correctly detects a LOW signal. Connect a jumper from the Raspberry Pi's GND pin to the breadboard's ground (-) rail.

2. **Place the Tactile Switches:** Tactile switches have four legs arranged in two electrically connected pairs. Mounting them across the central trench ensures that pressing the switch closes the circuit between those pairs. Orient each switch so that its two pairs of legs straddle the central trench of the breadboard. Verify that pressing the button connects the two legs on one side to the two legs on the opposite side.

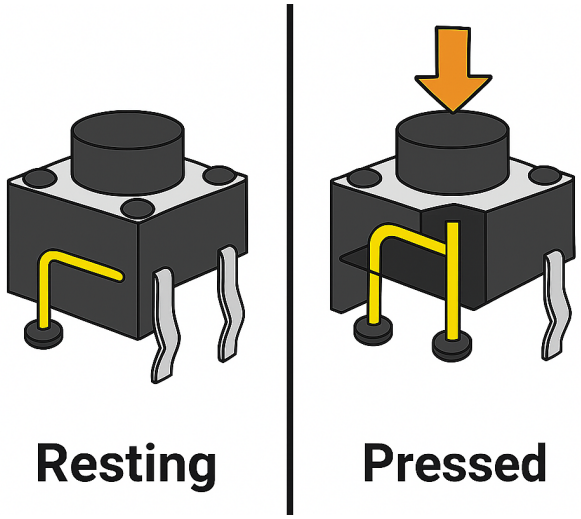

Figure 2: Circuit Diagram.

3. **Wire the Buttons:** Each button input uses one GPIO line and the ground rail. When the button is not pressed, the internal pull-up holds the GPIO pin at a logical HIGH. Pressing the button connects it to ground, producing a logical LOW that the program can detect.

- Connect one leg of each button to the ground rail (-) on the breadboard.
- Connect the other leg of Button 1 to GPIO 26 (physical pin 37 on the Pi).
- Connect the other leg of Button 2 to GPIO 20 (physical pin 38).
- Connect the other leg of Button 3 to GPIO 21 (physical pin 40).

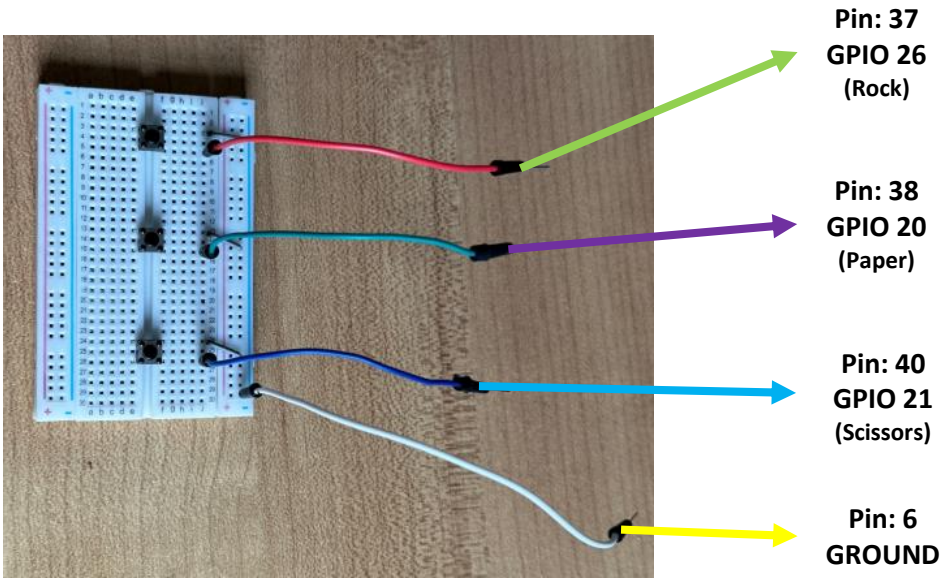

Figure 3: Buttons set-up.

LCD Display Wiring:

For the 20×4 character LCD, we assume it has an I²C interface adapter attached. This adapter greatly simplifies the wiring by using only four connections.

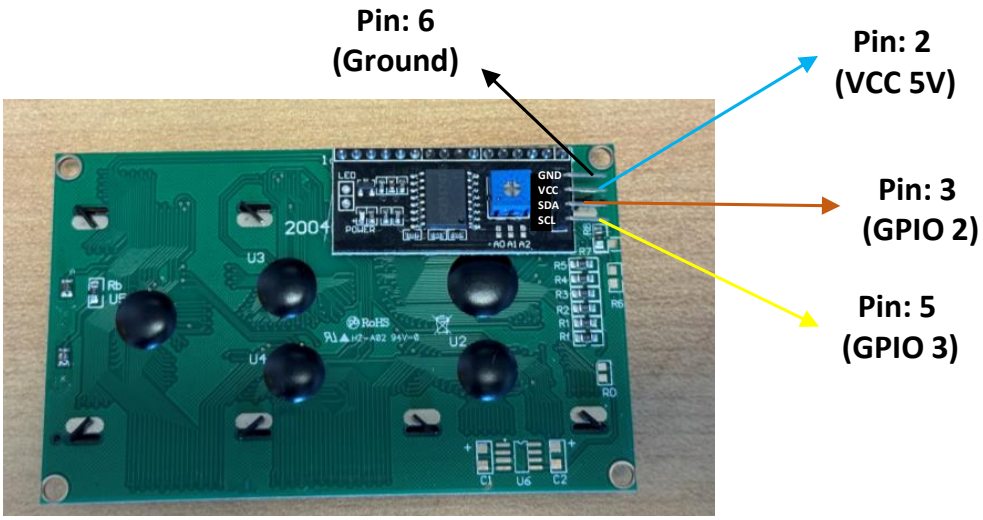

Figure 4: LCD connection to Raspberry Pi's GPIO pins.

1. Connect the LCD module's VCC pin to the Pi's 5V power (physical pin 2 or 4), and the LCD GND to a Pi GND pin 6.
2. Connect the LCD's SDA pin to the Pi's SDA line. On Raspberry Pi, SDA corresponds to GPIO 2 (physical pin 3)
3. Connect the LCD's SCL pin to the Pi's SCL line, which is GPIO 3 (physical pin 5).

Complete Setup

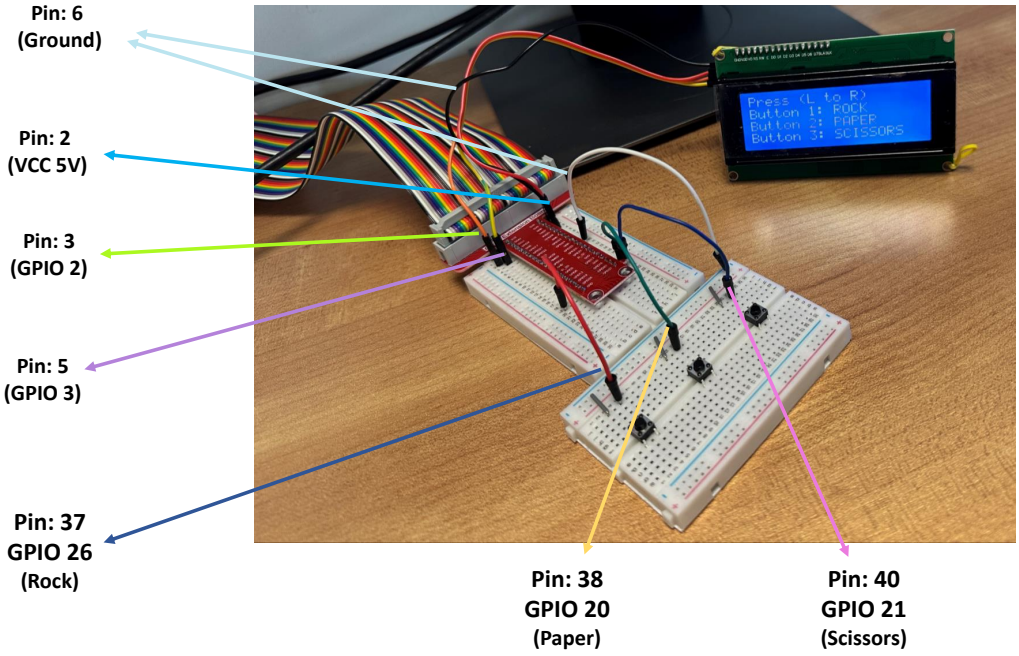

Figure 5: Wiring set-up.

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.

Record your observations:

Computer Choice	**User Choice**	**Result**

Table 1: Gane Results

Part 2: Enhanced Game Modes

In Part 2, students take the working single-round game from Part 1 and transform it into a clean, modular program by creating user-defined functions for each core task—such as reading a button press, displaying messages on the LCD, and determining the winner of a round. They then build three distinct game-mode loops (Sudden Death, Best of 3, and First to 3 Wins) that simply call these functions instead of retyping the same code, reinforcing how functions promote clarity, reduce duplication, and make complex programs easier to extend and maintain.

Tips for Implementation

- **Modularizing the core logic:** Students begin by defining focused functions—such as `get_choice()`, `display_message()`, and `decide_winner()`—each responsible for a single task. This separation makes the code more readable and easier to test or modify.
- **Eliminating repetition:** By invoking these functions wherever the associated behavior is needed, rather than copying-and-pasting code, students learn the DRY principle (Don't Repeat Yourself) and experience firsthand how refactoring reduces bugs and duplication.
- **Adding a mode-selection menu:** Before the game loop starts, the program displays an LCD menu letting the user choose between Sudden Death, Best of 3, or First to 3 Wins. This step introduces branch logic and menu navigation on the LCD.
- **Implementing Sudden Death mode:** In this simplest mode, the code calls a `play_round()` function exactly once and then immediately shows the winner. It illustrates how a single function call can encapsulate an entire round.
- **Implementing Best of 3 mode:** Students write a loop that repeatedly calls `play_round()` until either the user or the CPU wins two rounds, reinforcing loop constructs, condition checks, and state tracking of win counts.
- **Implementing First to 3 Wins mode:** This mode extends the loop logic to continue playing until a player reaches three wins, teaching students how to manage cumulative state and termination conditions over an indefinite number of rounds.
- **Incorporating LED enhancements:**
  - Encourage students to use the existing GPIO-connected LEDs as visual feedback—e.g., blink all three rapidly during mode selection, then light only the chosen mode's LED.
  - Define a helper function like `blink_led(led, times)` to signal round start/end or to indicate a win (three quick blinks) versus a loss (single long blink).
  - Use slow/fast blink patterns on the runner LEDs to reinforce the slowing animation in each game mode.
- **Displaying final results:** After the chosen mode completes, the program uses `display_message()` to show the final score tally and announce the overall winner on the LCD, demonstrating composition of functions for user feedback.

Reflection and Analysis

1. Why use functions? Explain how they improve readability and reduce duplication.
2. Mode logic: For each mode, outline in pseudocode how you check win counts.
3. LCD feedback: How does encapsulating display calls in a function simplify changes (e.g., adding animations)?
4. **Further extension:** Discuss how this project can be extended.