# Artifact 1 — CS 350: Smart Thermostat (Raspberry Pi)

## Brief Description (What/When)

This project implements a Raspberry Pi–based smart thermostat that integrates hardware sensors, buttons, LEDs, and an LCD display. The system reads temperature (AHT20 over I2C), allows mode switching (OFF/HEAT/COOL), and provides visual feedback with LEDs and an LCD. It logs telemetry data through UART for monitoring.  
  
Technologies: Raspberry Pi GPIO/I2C/UART, Python, LCD (16×2), LEDs, Buttons  
Core features: Sensor read, state machine control, LED PWM feedback, LCD updates, UART logging  
Original course: CS 350 – Emerging Systems Architectures and Technologies

## Why This Artifact? (Justification)

This artifact was chosen because it showcases embedded systems design, modular programming, and algorithmic control. It demonstrates how hardware and software integration requires careful design practices such as:  
- State machines for predictable control  
- Timers for logging and display switching  
- Debounced inputs for reliable button handling  
- Secure GPIO access for safety

## Enhancements Implemented

### 1) Software Design & Engineering

- Refactored into modular files (sensor.py, state.py, display.py, telemetry.py, io\_hw.py, app.py)  
- Added state machine abstraction for OFF, HEAT, and COOL modes  
- Created dedicated services for LCD output and UART logging  
- Improved error handling (I2C errors, GPIO cleanup, debounced button input)  
- Added testability via dependency injection (mock sensors and IO in unit tests)

### 2) Algorithms & Data Structures

- Implemented hysteresis control to avoid rapid toggling around setpoint  
- Used non-blocking timers for UART logging and LCD switching  
- Built deterministic finite-state logic for predictable transitions and LED PWM effects

## Reflection (Process, Challenges, Feedback)

What I learned: Designing embedded applications benefits from modular design, state machines, and input debouncing. Even small algorithmic improvements like hysteresis make the thermostat more professional.  
  
Challenges: Handling concurrent tasks (buttons, display, UART) without threads required careful non-blocking design.  
  
Feedback incorporated: Added inline comments, standardized variable names, and a safe-off routine to release GPIO pins.

## Outcomes Met

- Software Design & Engineering: modular, testable, readable Python code  
- Algorithms & Data Structures: hysteresis, timers, state machine  
- Security Mindset: validated inputs, safe GPIO release, no blocking IO  
- Professional Communication: documentation, pseudocode, and this narrative