SmartCycle Book

Consolidated Modules

SmartCycle Team

2025-12-15

Contents

Overview

```
---
title: Modules Overview
---

# Modules Overview

This chapter provides links and brief descriptions for each module.

- Module 1: Intro and main script
- Module 2: Components, error codes, and entrypoint
- Module 3: Flowchart and troubleshooting report
- Module 4: Planning documents
- Module 5: Readme notes
- Module 6: Readme notes

Use the sidebar to navigate to each detailed module page.
```

Module 1

Plan

```
# Module 1 — Plan

## 1. Data Model

- **Distance readings (`distance_readings`)**
  - Unit: miles
  - One element = total distance ridden in one day.
  - Typical daily value: around 29.2 miles.
  - Expected range: from about 20 miles to about 80 miles.

- **Battery readings (`battery_readings`)**
  - Unit: percent (%)
  - One element = average battery used (or remaining — specify) for one day.
  - Expected range: **40%–60%** per day.
  - I will treat values below 10% or above 90% as suspicious or "bad data."

- **Week structure**
  - Exactly **7 elements** in each list, one per day (Day 1–Day 7).
  - Day labels I will use in output: `D#` (e.g., Day 1–Day 7 or Mon–Sun).

## 2. Required Array Operations (Design)

I will demonstrate the following operations in my script.
Below I describe **where** each operation occurs in the program and **why** I use it.

1. **Insert**
   - Operation point: After initializing the weekly distance data.
   - Purpose: To correct missing ride data for one day.
   - Applied to: `distance_readings`
   - Scenario: The distance for Day 4 was not recorded due to a logging issue, so the correct value is inserted at the appropriate position in the list.

2. **Update**
   - Operation point: After all distance data has been verified and inserted.
   - Purpose: To correct an incorrect battery percentage reading.
   - Applied to: `battery_readings`
   - Scenario: A battery reading was logged outside the expected 40–60% range, so the value is updated to the correct percentage for that day.

3. **Delete / Pop**
   - Operation point: After reviewing all battery readings.
   - Purpose: To remove a corrupted or incomplete battery entry.
   - Applied to: `battery_readings`
   - Scenario: The final day's battery data was corrupted due to an unexpected device shutdown, so the last battery reading is removed from the list.

4. **Search**
   - Operation point: After all distance corrections have been completed.
   - Purpose: To determine whether any day exceeded a high-distance threshold.
   - Applied to: `distance_readings`
   - Scenario: The program checks if any daily distance is greater than 60 miles to identify unusually long rides.
```

5. **Slice**
   - Operation point: After validating the full week of distance data.
   - Purpose: To analyze riding patterns toward the end of the week.
   - Applied to: `distance_readings`
   - Slice range: Day 5 through Day 7.
   - Scenario: The program extracts the last three days of distance data to examine end-of-week riding behavior.

6. **Concatenate**
   - Operation point: After both arrays have been cleaned (insert/update/pop complete).
   - Purpose: To build a combined weekly dataset for reporting.
   - Arrays combined: `distance_readings` + `battery_readings`
   - Scenario: The program combines both arrays into one list so the week's ride data can be output or processed as a single dataset.

3. Required Calculations

1. **Total Weekly Distance**
   - Calculated after distance corrections are complete.
   - Uses: all 7 daily distance values.
   - Output: a single total in miles.

2. **Average Battery Percentage**
   - Calculated after battery corrections are complete.
   - Uses: the cleaned battery list values.
   - Output: a single average percentage for the week.

4. ASCII Bar Graph Design

- Metric visualized: **Daily distance travelled**
- Scale: **1 `#` = 5 miles**
- Maximum expected bars:
  - 80 miles = 16 `#` characters
- Format per line:
  - `Day X: ####...`
- Purpose:
  - To provide a simple, readable visual comparison of daily riding distances across the week.

Code: main.py

```
print("Welcome to the Module 1 Smartcycle Data Processing Application!")

# init values
distances = [24.5, 31.2, 27.8, 45.6, 62.4, 38.9]
battery = [48, 55, 61, 52, 47, 59, "24"]

if len(distances) != 7:
    print("Distance data is incomplete.")
    missed_day = int(input("Which day is missing distance data? Use a number (e.g. 1, 2, 3, 4,
5, 6, 7) > "))
    new_distance = float(input(f"Please enter the distance for day {missed_day} > "))
    distances.insert(missed_day - 1, new_distance)

if len(battery) != 7:
    print("Battery data is incomplete.")
    missed_day = int(input("Which day is missing battery data? Use a number (e.g. 1, 2, 3, 4, 5,
```

```
6, 7) > "))
    new_battery = int(input(f"Please enter the battery percentage for day {missed_day} > "))
    battery.insert(missed_day - 1, new_battery)

# clean corrupted distance entries before range checks
for day in range(len(distances)):
    if not isinstance(distances[day], (float, int)):
        print(f"Data for day {day + 1} seems off: Distance = {distances[day]}, Battery =
{battery[day]}")
        corrected_distance = float(input(f"Please enter the correct distance for day {day + 1} >
"))
        distances[day] = corrected_distance

# clean corrupted battery entries before range checks
for day in range(len(battery)):
    if not isinstance(battery[day], int):
        print(f"Data for day {day + 1} seems off: Distance = {distances[day]}, Battery =
{battery[day]}")
        corrected_battery = int(input(f"Please enter the correct battery percentage for day {day
+ 1} > "))
        battery[day] = corrected_battery

# validate distance ranges and let the user correct bad values
for day in range(len(distances)):
    if 20 <= distances[day] <= 80:
        continue

    print(f"Data for day {day + 1} seems off: Distance = {distances[day]}, Battery =
{battery[day]}")
    corrected_distance = float(input(f"Please enter the correct distance for day {day + 1} > "))
    distances[day] = corrected_distance

# validate battery ranges and let the user correct bad values
for day in range(len(battery)):
    if 40 <= battery[day] <= 60:
        continue

    print(f"Data for day {day + 1} seems off: Distance = {distances[day]}, Battery =
{battery[day]}")
    corrected_battery = int(input(f"Please enter the correct battery percentage for day {day +
1} > "))
    battery[day] = corrected_battery

# search for any outliers in distance data (e.g., values below 20 or above 80) only dry-running
for day in range(len(distances)):
    if distances[day] < 20 or distances[day] > 80:
        print(f"Outlier detected for day {day + 1}: Distance = {distances[day]}")

# slide end of week data from day 5 to 7 and output a report based on driving
day5_to7_distances = distances[4:7]
day5_to7_battery = battery[4:7]
total_distance = sum(day5_to7_distances)
average_battery = sum(day5_to7_battery) / len(day5_to7_battery)
print(f"From day 5 to 7, the total distance driven was {total_distance} m.")
print(f"The average battery percentage from day 5 to 7 was {average_battery}%.")

# concatenate distance and battery data into a single report for the week
weekly_report = []
for day in range(len(distances)):
```

```
    weekly_report.append({
        "day": day + 1,
        "distance": distances[day],
        "battery": battery[day]
    })
print("Weekly Report:")
for entry in weekly_report:
    print(f"Day {entry['day']}: Distance = {entry['distance']} m, Battery = {entry['battery']}")

# display an ascii bar graph
# each day 1 '#' is equal to 5 miles.
print("Distance Bar graph for each day:")
print("Each '#' represents 5 miles.")
for day in range(len(distances)):
    hashes = int(distances[day] / 5)
    print(f"Day {day + 1}: " + "#" * hashes)
    print("\n")

print("end of report.")
print("This is the end of the Module 1 program.")
```

Module 2

Plan

```
# Module 2 — Plan (Data Structures)

## 1. Dictionary: Component Inventory
**Structure:** Dictionary (dict)
**Purpose:** Store SmartCycle components so the system can quickly look up parts and update
quantities.

- **Key:** Component ID (e.g., "BAT001", "SEN102")
- **Value:** A small record containing:
  - Component name
  - Quantity available

**Operations I will demonstrate:**
- Look up a component using its ID
- Update the quantity after a component replacement
- Add a new component to the inventory

## 2. Set: Error Codes
**Structure:** Set
**Purpose:** Store unique error codes detected during SmartCycle rides.

- **Error code format:** Numeric-style strings (e.g., "E101", "E202", "E305")

**Operations I will demonstrate:**
- Look up error codes
- Clean up error code list (remove duplicates)

## 3. List: User Activity Logs
**Structure:** List
**Purpose:** Store a chronological record of user activity within the SmartCycle application.

- **Each log entry contains:**
  - User identifier
  - Action performed (e.g., "viewed weekly report", "synced device")
  - Timestamp of the action

**Operations I will demonstrate:**
- Append a new activity log entry
- Display recent user actions in order
- Scan the log to find when a specific action occurred

**Implementation:** Use a decorator pattern to automatically log every function call in main.py.
Each time a decorated function runs, it appends an entry to `activity_logs` with the function
name, timestamp, and user ID. Helper functions allow viewing recent logs and searching by action
keyword.

## 4. Justification Summary

- **Dictionary:**
  The component inventory uses a dictionary because each component can be uniquely identified by
its component ID. This allows for fast lookups, easy updates to quantities, and scalable
management as more components are added.
```

- **Set:**
  Error codes are stored in a set because only unique error codes are meaningful for diagnostics. Using a set automatically removes duplicates and allows quick checks to determine whether a specific error occurred.

- **List:**
  User activity logs are stored in a list because order matters. Lists preserve the sequence of events and allow duplicate actions, which is necessary when tracking repeated user behavior over time.

Components

```
# Init storage
# we will store smartcycle components in a dictionary so the system can quick look them up
# the dictionary will have component ID as the key such as BAT001, or SEN102, value will contain component name and how much are available.
smartcycle_components = {
    "RST001": {"name": "Carbon-Film Fixed Resistor 5% 1kΩ", "quantity": 45},
    "RST002": {"name": "Metal Film Fixed Resistor 1% 10kΩ", "quantity": 82},
    "RST003": {"name": "Metal Oxide Film Fixed Resistor 5% 2.2kΩ", "quantity": 23},
    "RST004": {"name": "Wire Wound Fixed Resistor 5% 100Ω", "quantity": 67},
    "RST005": {"name": "Thick Film SMD Fixed Resistor 5% 4.7kΩ", "quantity": 91},
    "RST006": {"name": "Thin Film SMD Fixed Resistor 1% 100kΩ", "quantity": 8},
    "RST007": {"name": "Fusible Fixed Resistor 5% 10Ω", "quantity": 0},
    "RST008": {"name": "Linear Potentiometer 10% 10kΩ", "quantity": 56},
    "RST009": {"name": "Precision Linear Resistor 1% 470Ω", "quantity": 34},
    "BMS001": {"name": "TI bq76952 16-Cell Battery Monitor - High accuracy voltage/temp monitoring for Li-ion packs", "quantity": 10},
    "BMS002": {"name": "Maxim MAX17320 ModelGauge m5 - Smart battery fuel gauge with protector", "quantity": 20},
    "BMS003": {"name": "Analog Devices LTC6811 Multi-Cell Monitor - 12-cell battery stack monitor IC", "quantity": 15},
    "BMS004": {"name": "Infineon TLE9012AQU - 12-channel battery cell controller with integrated balancing", "quantity": 5},
    "BMS005": {"name": "NXP MC33771B 14-Cell Monitor - Automotive grade battery management controller", "quantity": 3},
    "BMS006": {"name": "Renesas ISL94202 - Multi-cell Li-ion battery pack monitor with FET control", "quantity": 10},
    "GPS001": {"name": "u-blox NEO-M9N - Multi-GNSS receiver with concurrent GPS/GLONASS/Galileo/BeiDou", "quantity": 42},
    "GPS002": {"name": "Quectel L76-L - Ultra-compact GPS/GLONASS module with integrated antenna", "quantity": 67},
    "GPS003": {"name": "MediaTek MT3333 - High sensitivity GPS chipset with 99 channels", "quantity": 28},
    "GPS004": {"name": "STMicroelectronics Teseo-LIV3F - Tiny GNSS module with dead reckoning", "quantity": 15},
    "GPS005": {"name": "u-blox ZED-F9P - Multi-band RTK GNSS receiver with cm-level accuracy", "quantity": 8},
    "GPS006": {"name": "SkyTraq Venus638FLPx - Low power GPS receiver with 1Hz-10Hz update rate", "quantity": 53},
    "GPS007": {"name": "Trimble BD970 - Dual-frequency RTK GNSS OEM board for precision positioning", "quantity": 3},
}


def lookup(component_id):
    """Return component details by ID or prompt to add it if missing."""
```

```
    if component_id in smartcycle_components:
        entry = smartcycle_components[component_id]
        return f"ID: {component_id}, Name: {entry['name']}, Quantity: {entry['quantity']}"

    add_component = input(
        f"Component ID {component_id} not found. Would you like to add it to the SmartCycle
component system? (y/n): > "
    )
    if add_component.lower() == "y":
        component_name = input("Enter the component name: > ")
        component_quantity = int(input("Enter the component quantity: > "))
        smartcycle_components[component_id] = {"name": component_name, "quantity":
component_quantity}
        return smartcycle_components[component_id]

    return None


def update(component_id):
    """Interactively update a component, summarizing changes before apply."""
    if component_id not in smartcycle_components:
        return f"Component ID {component_id} not found."

    new_ID = input("Enter the new component ID, or press Enter to keep the current ID: > ")
    new_name = input("Enter the new component name, or press Enter to keep the current name: >
")
    new_quantity = input("Enter the new component quantity, or press Enter to keep the current
quantity: > ")

    if new_ID:
        if not (len(new_ID) == 6 and new_ID[0:3].isalpha() and new_ID[3:6].isdigit()):
            return "Invalid component ID format. It should be in the format AAA###."
        if new_ID in smartcycle_components and new_ID != component_id:
            return "Component ID already exists."

    current_entry = smartcycle_components[component_id]
    updated_entry = current_entry.copy()
    summary_lines = []

    if new_ID:
        summary_lines.append(f"ID: {component_id} -> {new_ID}")
    else:
        summary_lines.append(f"ID: unchanged ({component_id})")

    if new_name:
        updated_entry["name"] = new_name
        summary_lines.append(f"Name: '{current_entry['name']}' -> '{new_name}'")
    else:
        summary_lines.append(f"Name: unchanged ('{current_entry['name']}')")

    if new_quantity:
        try:
            proposed_qty = int(new_quantity)
        except ValueError:
            return "Invalid quantity. Please enter a number."
        updated_entry["quantity"] = proposed_qty
        summary_lines.append(f"Quantity: {current_entry['quantity']} -> {proposed_qty}")
    else:
        summary_lines.append(f"Quantity: unchanged ({current_entry['quantity']})")
```

```
    print("Summary of changes:")
    for line in summary_lines:
        print(f"- {line}")

    confirm = input("Apply these changes? (y/n): > ").strip().lower()
    if confirm != "y":
        return "Update cancelled."

    if new_ID and new_ID != component_id:
        smartcycle_components[new_ID] = updated_entry
        del smartcycle_components[component_id]
        component_id = new_ID
    else:
        smartcycle_components[component_id] = updated_entry

    return f"Updated {component_id}: {updated_entry['name']} (qty {updated_entry['quantity']})"


def add(component_id, component_name, component_quantity):
    """Add a new component, prompting for missing fields."""
    if component_id in smartcycle_components:
        return f"Component ID {component_id} already exists."
    if not (len(component_id) == 6 and component_id[0:3].isalpha() and
component_id[3:6].isdigit()):
        return "Invalid component ID format. It should be in the format AAA###."
    if not component_name:
        component_name = input("Enter the component name: > ")
    if not component_quantity:
        component_quantity = int(input("Enter the component quantity: > "))
    smartcycle_components[component_id] = {"name": component_name, "quantity":
component_quantity}

    return smartcycle_components[component_id]


def delete(component_id):
    """Delete a component after confirmation."""
    if component_id not in smartcycle_components:
        return f"Component ID {component_id} not found."
    confirm = input(f"are you sure you would like to delete component {component_id}? (y/n): >
")
    confirm = confirm.strip().lower()
    if confirm != "y":
        return "Deletion cancelled."

    del smartcycle_components[component_id]
    return f"Component {component_id} has been successfully deleted."
```

Error Codes

```
# Error codes. Smartcycle components have unique sensors on each component. If a component on
each component fails it will trigger an error codes. However due to the variety of components,
some error codes are shared between components and some are unique to a specific component.
from tabulate import tabulate
error_codes = {
    "E101": {"description": "Over-voltage detected", "components": ["BMS001", "BMS002",
"BMS003", "BMS004", "BMS005", "BMS006"]},
```

```
    "E102": {"description": "Under-voltage detected", "components": ["BMS001", "BMS002",
"BMS003", "BMS004", "BMS005", "BMS006"]},
    "E103": {"description": "Cell temperature high", "components": ["BMS001", "BMS002",
"BMS003", "BMS004", "BMS005", "BMS006"]},
    "E104": {"description": "Cell temperature low", "components": ["BMS001", "BMS002", "BMS003",
"BMS004", "BMS005", "BMS006"]},
    "E105": {"description": "Pack current over limit", "components": ["BMS001", "BMS002",
"BMS003", "BMS004", "BMS005", "BMS006"]},
    "E106": {"description": "Short circuit detected", "components": ["BMS001", "BMS002",
"BMS003", "BMS004", "BMS005", "BMS006"]},
    "E107": {"description": "Cell imbalance above threshold", "components": ["BMS001", "BMS003",
"BMS004", "BMS005"]},
    "E108": {"description": "Charge FET stuck on", "components": ["BMS002", "BMS006"]},
    "E109": {"description": "Discharge FET stuck on", "components": ["BMS002", "BMS006"]},
    "E110": {"description": "State-of-charge estimation fault", "components": ["BMS001",
"BMS002", "BMS003"]},
    "E111": {"description": "Internal ADC fault", "components": ["BMS001", "BMS003", "BMS004"]},
    "E112": {"description": "EEPROM configuration error", "components": ["BMS002", "BMS005"]},
    "E113": {"description": "Balancing transistor overheat", "components": ["BMS004",
"BMS005"]},
    "E114": {"description": "Pack harness disconnect", "components": ["BMS001", "BMS003",
"BMS005"]},
    "E115": {"description": "Isolation fault to chassis", "components": ["BMS004", "BMS006"]},
    "E116": {"description": "Precharge failure", "components": ["BMS001", "BMS003", "BMS006"]},
    "E117": {"description": "Fuse blown", "components": ["BMS004", "BMS005", "BMS006"]},
    "E118": {"description": "Pack contactor welded", "components": ["BMS004", "BMS006"]},
    "E119": {"description": "Pack contactor open", "components": ["BMS004", "BMS006"]},
    "E120": {"description": "Cell voltage sensor fault", "components": ["BMS001", "BMS003",
"BMS005"]},
    "E121": {"description": "Thermistor open", "components": ["BMS001", "BMS002", "BMS003",
"BMS006"]},
    "E122": {"description": "Thermistor short", "components": ["BMS001", "BMS002", "BMS003",
"BMS006"]},

    "E201": {"description": "GNSS antenna open", "components": ["GPS001", "GPS002", "GPS003",
"GPS004", "GPS005", "GPS006", "GPS007"]},
    "E202": {"description": "GNSS antenna short", "components": ["GPS001", "GPS002", "GPS003",
"GPS004", "GPS005", "GPS006", "GPS007"]},
    "E203": {"description": "No satellite lock", "components": ["GPS001", "GPS002", "GPS003",
"GPS004", "GPS005", "GPS006", "GPS007"]},
    "E204": {"description": "Low signal-to-noise ratio", "components": ["GPS001", "GPS002",
"GPS003", "GPS004", "GPS005", "GPS006", "GPS007"]},
    "E205": {"description": "Time-to-first-fix timeout", "components": ["GPS001", "GPS002",
"GPS003", "GPS004", "GPS005", "GPS006", "GPS007"]},
    "E206": {"description": "Ephemeris download failure", "components": ["GPS001", "GPS002",
"GPS003", "GPS004", "GPS005", "GPS006", "GPS007"]},
    "E207": {"description": "Antenna detuned", "components": ["GPS001", "GPS002", "GPS003",
"GPS004", "GPS005", "GPS006"]},
    "E208": {"description": "RTC backup battery low", "components": ["GPS001", "GPS002",
"GPS003", "GPS004"]},
    "E209": {"description": "PPS output missing", "components": ["GPS005", "GPS006", "GPS007"]},
    "E210": {"description": "RTK correction timeout", "components": ["GPS005", "GPS007"]},
    "E211": {"description": "Multipath suspected", "components": ["GPS001", "GPS002", "GPS003",
"GPS004", "GPS005", "GPS006", "GPS007"]},
    "E212": {"description": "IMU fusion fault", "components": ["GPS004", "GPS005"]},
    "E213": {"description": "Dead-reckoning drift high", "components": ["GPS004", "GPS005"]},
    "E214": {"description": "Firmware image corrupt", "components": ["GPS001", "GPS003",
"GPS004", "GPS006"]},
    "E215": {"description": "GNSS L1 band blocked", "components": ["GPS001", "GPS002", "GPS003",
```

```
"GPS004", "GPS005", "GPS006", "GPS007"]},
    "E216": {"description": "GNSS L2 band blocked", "components": ["GPS005", "GPS007"]},
    "E217": {"description": "Temperature out of range", "components": ["GPS001", "GPS002",
"GPS003", "GPS004", "GPS005", "GPS006", "GPS007"]},
    "E218": {"description": "UART communication fault", "components": ["GPS001", "GPS002",
"GPS003", "GPS004", "GPS005", "GPS006", "GPS007"]},
    "E219": {"description": "I2C communication fault", "components": ["GPS002", "GPS003",
"GPS004", "GPS005"]},
    "E220": {"description": "SPI communication fault", "components": ["GPS003", "GPS004",
"GPS005", "GPS006"]},
    "E221": {"description": "Data flash wearout warning", "components": ["GPS001", "GPS004",
"GPS006"]},
    "E222": {"description": "Jamming detected", "components": ["GPS001", "GPS002", "GPS003",
"GPS004", "GPS005", "GPS006", "GPS007"]},

    "E301": {"description": "Resistor open circuit", "components": ["RST001", "RST002",
"RST003", "RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E302": {"description": "Resistor drift beyond tolerance", "components": ["RST001",
"RST002", "RST003", "RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E303": {"description": "Resistor overheating", "components": ["RST001", "RST002", "RST003",
"RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E304": {"description": "Lead corrosion detected", "components": ["RST001", "RST003",
"RST004", "RST005", "RST007"]},
    "E305": {"description": "Solder joint crack", "components": ["RST001", "RST002", "RST003",
"RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E306": {"description": "ESD damage suspected", "components": ["RST002", "RST006", "RST008",
"RST009"]},
    "E307": {"description": "Moisture ingress", "components": ["RST003", "RST004", "RST005",
"RST007"]},
    "E308": {"description": "Potentiometer wiper noise", "components": ["RST008"]},
    "E309": {"description": "Fusible link blown", "components": ["RST007"]},
    "E310": {"description": "Wirewound hot spot", "components": ["RST004"]},
    "E311": {"description": "Thin film drift", "components": ["RST006"]},
    "E312": {"description": "Thick film drift", "components": ["RST005"]},
    "E313": {"description": "Carbon film noise", "components": ["RST001"]},
    "E314": {"description": "Metal film noise", "components": ["RST002", "RST009"]},
    "E315": {"description": "Oxide film value shift", "components": ["RST003"]},
    "E316": {"description": "Temperature coefficient exceeded", "components": ["RST002",
"RST004", "RST006", "RST009"]},
    "E317": {"description": "Mechanical vibration loosened lead", "components": ["RST001",
"RST003", "RST004", "RST005", "RST007"]},
    "E318": {"description": "SMD tombstoning", "components": ["RST005", "RST006"]},
    "E319": {"description": "Lead wire fatigue", "components": ["RST001", "RST003", "RST004",
"RST007"]},
    "E320": {"description": "Overpower condition", "components": ["RST001", "RST002", "RST003",
"RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E321": {"description": "Hotspot detected near pad", "components": ["RST005", "RST006"]},
    "E322": {"description": "Wiper end-stop overrun", "components": ["RST008"]},
    "E323": {"description": "Precision tolerance exceeded", "components": ["RST002", "RST009"]},
    "E324": {"description": "Strain-induced value shift", "components": ["RST001", "RST003",
"RST004"]},
    "E325": {"description": "Overvoltage surge", "components": ["RST001", "RST002", "RST003",
"RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E326": {"description": "Thermal runaway risk", "components": ["RST003", "RST004", "RST005",
"RST006", "RST007"]},
    "E327": {"description": "Coating delamination", "components": ["RST003", "RST004",
"RST005"]},
    "E328": {"description": "Core saturation (wirewound)", "components": ["RST004"]},
    "E329": {"description": "Noise floor elevated", "components": ["RST001", "RST002", "RST003",
```

```
"RST009"]},
    "E330": {"description": "Contact resistance high", "components": ["RST008"]},
    "E331": {"description": "Lead short to chassis", "components": ["RST001", "RST003",
"RST004", "RST005", "RST007"]},
    "E332": {"description": "Open via to resistor pad", "components": ["RST005", "RST006"]},
    "E333": {"description": "Laser trim drift", "components": ["RST002", "RST006", "RST009"]},
    "E334": {"description": "Value mismatch in parallel network", "components": ["RST001",
"RST002", "RST003", "RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E335": {"description": "Value mismatch in series network", "components": ["RST001",
"RST002", "RST003", "RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E336": {"description": "Potentiometer dead spot", "components": ["RST008"]},
    "E337": {"description": "Precision drift beyond 1%", "components": ["RST002", "RST009"]},
    "E338": {"description": "Fusible element ageing", "components": ["RST007"]},
    "E339": {"description": "Lead plating corrosion", "components": ["RST001", "RST003",
"RST004", "RST005", "RST007"]},
    "E340": {"description": "Pad lift from PCB", "components": ["RST005", "RST006"]},
    "E341": {"description": "Mechanical shock damage", "components": ["RST001", "RST003",
"RST004", "RST005", "RST007", "RST008"]},
    "E342": {"description": "Vibration-induced microcracks", "components": ["RST001", "RST003",
"RST004", "RST005", "RST007", "RST008"]},
    "E343": {"description": "High humidity resistance drift", "components": ["RST001", "RST003",
"RST005", "RST006"]},
    "E344": {"description": "Temperature cycling stress", "components": ["RST002", "RST004",
"RST006", "RST009"]},
    "E345": {"description": "Hot spot from adjacent component", "components": ["RST005",
"RST006"]},
    "E346": {"description": "Substrate microfracture", "components": ["RST002", "RST006",
"RST009"]},
    "E347": {"description": "Terminal oxidation", "components": ["RST001", "RST003", "RST004",
"RST005", "RST007"]},
    "E348": {"description": "PCB contamination leakage", "components": ["RST005", "RST006"]},
    "E349": {"description": "Overcleaning removed coating", "components": ["RST003", "RST004",
"RST005"]},
    "E350": {"description": "Calibration required", "components": ["RST008", "RST009"]},
    "E351": {"description": "Thermal coefficient mismatch in network", "components": ["RST001",
"RST002", "RST003", "RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E352": {"description": "Noise exceeds specification", "components": ["RST001", "RST002",
"RST003", "RST009"]},
    "E353": {"description": "Current noise in potentiometer", "components": ["RST008"]},
    "E354": {"description": "Contact lift-off detected", "components": ["RST004", "RST007"]},
    "E355": {"description": "Thermal fuse nuisance trip", "components": ["RST007"]},
    "E356": {"description": "Value shift after surge", "components": ["RST001", "RST002",
"RST003", "RST004", "RST005", "RST006", "RST007", "RST008", "RST009"]},
    "E357": {"description": "High-frequency self-resonance issue", "components": ["RST002",
"RST004", "RST006", "RST009"]},
    "E358": {"description": "Lead inductance too high", "components": ["RST004"]},
    "E359": {"description": "PCB pad voiding", "components": ["RST005", "RST006"]},
    "E360": {"description": "Potentiometer track wear", "components": ["RST008"]},
    "E361": {"description": "Value jump during adjustment", "components": ["RST008"]},
    "E362": {"description": "Precision resistor humidity drift", "components": ["RST009"]},
    "E363": {"description": "Metal film microcracks", "components": ["RST002", "RST009"]},
    "E364": {"description": "Wirewound vibration rattle", "components": ["RST004"]},
    "E365": {"description": "Fusible resistor temperature creep", "components": ["RST007"]},
    "E366": {"description": "Carbon film burn spot", "components": ["RST001"]},
}

# function to lookup multiple error codes from usage logs
def lookup_error_codes(error_codes_list=None):
    # paste in csv format error codes you want to lookup if error codes are not provided in the
```

```
function call
    if error_codes_list is None:
        error_codes_list = input("Enter the error codes you want to lookup (comma-separated): >
")
        error_codes_list = error_codes_list.split(",")
    for code in error_codes_list:
        # if error code does not follow the pattern E### then raise an error saying that it must
follow the apttern
        if not code.startswith("E") or len(code) != 5 or not code[1:].isdigit():
            raise ValueError(f"Error code {code} must folow the pattern E###")
        # if error code does not exist in the error codes list then also raise an error saying
that there exists no such error code
        if code not in error_codes:
            raise ValueError(f"Error code {code} does not exist in the error codes list")
        # print the description and components associated with the error code in a table format
we can use the tabulate library for this
        print(tabulate([[code, error_codes[code]['description'], ", ".join(error_codes[code]
['components'])]], headers=["Error Code", "Description", "Components"], tablefmt="grid"))

# this will be a funtion to do a sweep and automatically remove any duplicate error codes from
the list of error codes
def clean_error_codes():
    # paste in csv format error codes you want to lookup
    error_codes_list = input("Enter the error codes you want to lookup (comma-separated): > ")
    error_codes_list = error_codes_list.split(",")
    # remove any duplicate error codes from the list of error codes
    error_codes_list = list(set(error_codes_list))
    # print the cleaned list of error codes
    print("Cleaned list of error codes:")
    for code in error_codes_list:
        print(code)
    # at the descretion of the user, ask if they want the error codes to be looked up
    lookup = input("Do you want to look up the error codes? (y/n): > ")
    lookup = lookup.lower()
    if lookup == "y":
        lookup_error_codes(error_codes_list)
```

Main Script

```
from datetime import datetime
from components import smartcycle_components, lookup, update, add, delete
from error_codes import error_codes

# Activity logs list to store all function calls
activity_logs = []


def log_activity(user_id="system"):
    """Decorator to automatically log function calls to activity_logs."""
    def decorator(func):
        def wrapper(*args, **kwargs):
            action = f"called {func.__name__}"
            timestamp = datetime.now().isoformat()
            activity_logs.append({
                "user_id": user_id,
                "action": action,
                "timestamp": timestamp
            })
```

```
            return func(*args, **kwargs)
        return wrapper
    return decorator


def view_recent_logs(n=10):
    """Display the most recent n activity log entries in reverse chronological order."""
    if not activity_logs:
        print("No activity logs found.")
        return
    print(f"\nRecent {min(n, len(activity_logs))} activity logs:")
    for log in activity_logs[-n:][::-1]:
        print(f"{log['timestamp']} | User: {log['user_id']} | {log['action']}")


def find_action(action_keyword):
    """Search activity logs for entries containing a specific action keyword."""
    matches = [log for log in activity_logs if action_keyword.lower() in log['action'].lower()]
    if not matches:
        print(f"No logs found containing '{action_keyword}'.")
        return []
    print(f"\nFound {len(matches)} log entries with '{action_keyword}':")
    for log in matches:
        print(f"{log['timestamp']} | User: {log['user_id']} | {log['action']}")
    return matches


@log_activity()
def summarize_inventory():
    print(f"Loaded {len(smartcycle_components)} components.")
    print(f"Loaded {len(error_codes)} error codes.")


@log_activity()
def list_components():
    for cid, entry in smartcycle_components.items():
        print(f"{cid}: {entry['name']} (qty {entry['quantity']})")


@log_activity()
def list_error_codes():
    for code, info in error_codes.items():
        targets = ", ".join(info["components"])
        print(f"{code}: {info['description']} -> {targets}")


if __name__ == "__main__":
    summarize_inventory()
    print("\nUse lookup/add/update/delete for component operations.")
    print("Use list_error_codes() to review error mappings.")
    print("\nActivity Logging:")
    print("  - view_recent_logs(n=10) to see recent activity")
    print("  - find_action(keyword) to search logs")
```

Module 3

Troubleshooting Report

```
# Module 3 – Troubleshooting Report

## Scenario
SmartCycle is no longer recieving sensor data from their components.

## Step 1: Problem

- When users report issues, the SmartCycle support team cannot see sensor logs in the monitoring
application.
- The system log file has not been updated or written to for an extended period of time.

## Step 2: Possible Causes

Several potential causes could explain why SmartCycle is no longer receiving sensor data:

- One or more sensors may be damaged, disconnected, or powered off.
- Sensors may be unable to transmit data due to a broken wired connection or wireless
communication issue.
- A recent update or bug may have caused the data collection service to stop running.
- The service responsible for writing sensor data to log files may be stopped or misconfigured.
- Insufficient power or battery problems may prevent sensors from operating correctly.

## Step 3: Solutions

Based on the possible causes, the following solutions are possible:

- Inspect physical sensors and ensure all components are securely attached.
- Restart the sensor communication interface to re-establish data transmission.
- Check firmware and application versions and roll back or patch recent updates if necessary.
- Restart or repair the logging service responsible for recording sensor data.
- Verify power levels and replace or recharge batteries if needed.

## Step 4: Testing

- After checking the hardware, confirm whether sensor data begins appearing in the monitoring
application.
- Restart communication services and observe whether new log entries are generated.
- Apply software fixes and monitor system behavior for errors or warnings.
- After restarting the logging service, check timestamps on log files to confirm new data is
being written.
- Monitor sensor readings after addressing power issues to ensure stable operation.

## Step 5: Verification and Mitigation of this issue

Once the sensor data is restored, the system will be monitored over an extended period of
time.
Logs will be reviewed to ensure data is consistently recorded, and alerts will be configured to
notify the team if data transmission stops again.

## Step 6: Documentation

All troubleshooting steps, test results, and final resolutions will be documented.
```

```
This documentation will be used to improve future response times and prevent similar sensor data
failures from occurring again.
```

Flowchart

This file is an Excalidraw diagram (open separately): module3/flowchart.excalidraw

Module 4

Module 4 — Collaboration and Productivity Tools

Overview
This module documents how collaboration and productivity tools were used to plan, organize, and complete the SmartCycle Data Console capstone project. The tools supported communication, file sharing, feedback, and coordination throughout development.

Tools Used

Google Docs
Used to collaboratively plan project modules, document requirements, and leave feedback through comments and suggestions.

Google Drive
Used to store and organize all project files in a shared folder. Permissions were managed to ensure proper access for collaborators.

Google Meet
Used for virtual meetings to discuss project progress, troubleshoot issues, and review module requirements.

Google Groups
Used to centralize project-related communication and announcements for team members.

Collaboration Workflow

1. Project planning and module outlines were created in Google Docs.
2. All documents and assets were stored in a shared Google Drive folder.
3. Team members used comments and @mentions in Google Docs to provide feedback.
4. Meetings were held using Google Meet to discuss progress and resolve issues.
5. Google Groups was used to send updates and reminders to the team.

Evidence of Collaboration

The following evidence demonstrates effective collaboration and tool usage:
- Shared Google Docs with comments and suggestions
- Google Drive folder with shared permissions
- Google Meet meeting records or calendar invites
- Google Groups discussion or group overview

Module 5

```
# Module 5 – Intellectual Property and Innovation

## Overview
This module identifies the intellectual property (IP) associated with the SmartCycle Data
Console project and explains how different forms of IP protection apply. The goal is to protect
innovation, prevent misuse, and preserve the value of the system.



## Types of Intellectual Property

### Patents
Patents protect new and useful inventions or processes.

**Applicable SmartCycle Elements:**
- Any novel method used to process or analyze sensor data
- Unique algorithms that optimize performance or detect anomalies
- Custom hardware–software integration methods

**Why Patents Matter:**
Patents prevent other companies from copying innovative technical solutions and allow the
inventor to control how the technology is used or licensed.



### Copyright
Copyright protects original creative works.

**Applicable SmartCycle Elements:**
- Source code written for data processing and reporting
- User interface designs and layouts
- Documentation, reports, and user guides

**Why Copyright Matters:**
Copyright ensures that original software and written materials cannot be copied or distributed
without permission.



### Trademarks
Trademarks protect brand identity.

**Applicable SmartCycle Elements:**
- The name “SmartCycle Data Console”
- Logos, icons, and branding elements

**Why Trademarks Matter:**
Trademarks prevent confusion in the marketplace and protect the reputation and identity of the
product.



## IP Mapping Summary

| SmartCycle Component            | IP Type       | Reason |
|---------------------------------|---------------|--------|
```

Sensor data processing logic	Patent	Novel technical process
Application source code	Copyright	Original software work
UI design and documentation	Copyright	Creative expression
Product name and branding	Trademark	Brand identity protection

Risks of Not Protecting IP

Failing to protect intellectual property could result in:
- Competitors copying key features or designs
- Loss of brand identity
- Reduced ability to monetize or license the technology
- Legal disputes over ownership

Connection to Innovation Protection Practices

This approach aligns with real-world innovation protection strategies, such as those outlined in professional engineering and entrepreneurship toolkits. Identifying what is innovative and applying the correct form of IP protection helps ensure long-term sustainability and ethical use of technology.

Module 6

Module 6 — Data Privacy and Ethics

Overview
This module examines the data collected by the SmartCycle Data Console and evaluates the privacy and ethical implications of handling that data. It also proposes safeguards to ensure responsible and ethical use of user information.

Data Collected by SmartCycle

The SmartCycle system may collect the following types of data:

- Ride distance and duration
- Battery usage statistics
- Sensor performance data
- Device identifiers
- User account information (e.g., username or email)

This data is used to improve performance monitoring, diagnostics, and user experience.

Privacy Risks

Improper handling of SmartCycle data could introduce several privacy risks:

- **Unauthorized tracking:** Ride data could reveal user routines or locations.
- **Data breaches:** Stored data could be exposed through hacking or system vulnerabilities.
- **Secondary data misuse:** Data could be used for purposes not originally disclosed to users.

These risks highlight the importance of transparency and strict access controls.

Ethical Concerns and Real-World Context

Historical cases such as the Cambridge Analytica scandal demonstrate how personal data can be exploited when collected without clear consent or used beyond its intended purpose.

To avoid similar ethical failures, SmartCycle must:
- Clearly explain what data is collected and why
- Avoid collecting unnecessary personal information
- Prevent data from being shared without user consent

Data Protection Strategies

SmartCycle can protect user data through the following measures:

- **Data minimization:** Only collect data required for system functionality.
- **User consent:** Clearly request permission before collecting or storing data.
- **Secure storage:** Encrypt stored data and restrict access to authorized systems.
- **Retention limits:** Automatically delete data after it is no longer needed.
- **User control:** Allow users to view, export, or delete their data.

Ethical Impact Assessment

Positive Impacts
- Improved system reliability
- Better diagnostics and maintenance
- Enhanced user experience

Potential Negative Impacts
- Loss of privacy if data is mishandled
- User mistrust if data practices are unclear
- Legal and ethical consequences from misuse

Responsible data governance ensures that the benefits of SmartCycle outweigh potential harms.

Conclusion

Ethical data handling and strong privacy protections are essential to maintaining trust and compliance. By implementing clear policies and technical safeguards, SmartCycle can responsibly manage user data while delivering meaningful value.