PROGRAMMING FUNDAMENTALS

Mission Control Panel v7.0

Week 7 Workshop • Working with Strings and Text Data

Dr Lei Wang (Griffith University)

The story
v6.0 taught Mission Control to store structured sensor data using lists and tuples.

This week we keep that architecture and improve how the system handles and communicates text using Week 7 string concepts.

Week 7 Mission

Start from your working v6.0 program. This week we add a stronger text interface and clearer formatted output.

- Use strip(), lower(), upper() and title() to clean and normalise text.
- Use indexing and slicing to extract parts of strings.
- Use in and find()-style searching where appropriate.
- Use split() and join() to move between text and collections of words/lines.
- Use f-strings, alignment and decimal precision for clearer output.
- Keep the v6.0 lists, tuples, loops, functions, filtering and analysis.

Core idea
Week 6 organised Mission Control's DATA.

Week 7 improves how Mission Control WORKS WITH and COMMUNICATES that data as text.

Prepare Your Files

1. Keep your working v6.0 file as mission_control_v6.py.
2. Create mission_control_v7.py.
3. Run v6.0 first and confirm scan, report, history, recent, safe, warning, critical, top3 and shutdown work.
4. Make one change at a time and test after each change.

Mission 1 — Add Mission Identity

Use string input, strip(), title() and an f-string to make the dashboard more readable.

```
mission_name = input("Mission name: ").strip().title()
destination = input("Destination: ").strip().title()

print("=" * 60)
print(f"MISSION CONTROL v7.0 — {mission_name}".center(60))
```

```
print(f"Destination: {destination}".center(60))
print("=" * 60)
```

Try: entering ' lunar survey ' and ' moon '. The stored/displayed text should be cleaned.

Mission 2 — Create a Reusable Reading Formatter

v6.0 repeatedly prints sensor number, temperature and status. Create one function that returns a formatted string.

```
def format_reading(reading):
    sensor_number, temperature, status = reading
    return f"Sensor {sensor_number:<2} | {temperature:>6.1f} C | {status:<8}"
```

This combines tuple unpacking with Week 7 f-string formatting and alignment.

```
reading = (3, 72.5, "SAFE")
print(format_reading(reading))
```

Expected output

Sensor 3 | 72.5 C | SAFE

Mission 3 — Upgrade HISTORY with join()

Build a list of formatted lines, then combine them into one multi-line string.

```
def show_history(reading_history):
    print("\n--- READING HISTORY ---")

    if not reading_history:
        print("No valid sensor readings recorded.")
        return

    lines = []
    for reading in reading_history:
        lines.append(format_reading(reading))

    print("\n".join(lines))
```

Key idea

'\n'.join(lines) puts each item on a new line. The join() method belongs to the separator string.

Mission 4 — Make RECENT Explicitly Use Slicing

```
def show_recent(reading_history):
    print("\n--- RECENT READINGS ---")

    recent_readings = reading_history[-3:]

    if not recent_readings:
        print("No valid sensor readings recorded.")
        return

    for reading in recent_readings:
        print(format_reading(reading))
```

reading_history[-3:] means the last three readings, or all available readings if fewer than three exist.

Mission 5 — Add a HELP Command with split() and join()

```
def show_help():
    command_text = (
        "scan report history recent safe warning critical "
        "top3 mission search status log help shutdown"
    )

    commands = command_text.split()

    print("\n--- AVAILABLE COMMANDS ---")
    print(" | ".join(commands))
```

Here we move from one string to a list with split(), then from the list back to a formatted string with join().

Mission 6 — Add a MISSION Command

```
def show_mission(mission_name, destination):
    print("\n--- MISSION INFORMATION ---")
    print(f"Mission: {mission_name}")
    print(f"Destination: {destination}")

    if len(mission_name) >= 3:
        print(f"Mission code: {mission_name[:3].upper()}")
    else:
        print("Mission code: " + mission_name.upper())
```

mission_name[:3] selects the first three characters. upper() then converts them to uppercase.

Mission 7 — Add Text SEARCH

```
def search_history(reading_history):
    keyword = input("Search text: ").strip().lower()

    if keyword == "":
        print("Search text cannot be empty.")
        return

    matches = []

    for reading in reading_history:
        line = format_reading(reading)

        if keyword in line.lower():
            matches.append(line)

    print("\n--- SEARCH RESULTS ---")

    if not matches:
        print("No matching readings found.")
    else:
        print("\n".join(matches))
```

The search is case-insensitive because both the keyword and the formatted line are converted to lowercase.

Mission 8 — Add Flexible STATUS Input

```
def status_command(reading_history):
    status = input(
        "Enter status (SAFE/WARNING/CRITICAL): "
    ).strip().upper()

    if status in ("SAFE", "WARNING", "CRITICAL"):
        show_status(reading_history, status)
    else:
        print("Invalid status.")
```

strip() removes surrounding spaces, upper() normalises case, and in checks whether the text is an allowed status.

Mission 9 — Build a Text Log

```
def show_log(reading_history):
    print("\n--- MISSION LOG ---")

    if not reading_history:
        print("No valid sensor readings recorded.")
        return

    lines = []

    for reading in reading_history:
        lines.append(format_reading(reading))

    log = "\n".join(lines)
    print(log)
    print(f"\nLog contains {len(lines)} readings.")
```

Mission 10 — Upgrade the Main Command Loop

Keep the v6.0 while loop and existing commands. Add the new Week 7 text-oriented commands.

```
command = input(
    "\nCommand [scan/report/history/recent/safe/warning/"
    "critical/top3/mission/search/status/log/help/shutdown]: "
).strip().lower()

if command == "mission":
    show_mission(mission_name, destination)
    continue

if command == "search":
    search_history(reading_history)
    continue

if command == "status":
    status_command(reading_history)
    continue

if command == "log":
    show_log(reading_history)
    continue

if command == "help":
```

```
        show_help()
        continue
```

The remaining v6.0 command handling stays in place.

Full Mission Control v7.0 — Complete Reference Code

Use this after building

Try the missions first.

Then compare your version with this complete reference implementation.

```
# ============================================================
# MISSION CONTROL v7.0
# Week 7: Strings and Text Data
# Based on Mission Control v6.0
# ============================================================

def show_header(title):
    print("=" * 60)
    print(title.center(60))
    print("=" * 60)

def classify_temperature(temperature):
    if temperature < 0:
        return "INVALID"
    elif temperature >= 100:
        return "CRITICAL"
    elif temperature > 80:
        return "WARNING"
    else:
        return "SAFE"

def scan_sensors():
    scan_readings = []
    safe_count = 0
    warning_count = 0

    for sensor_number in range(1, 6):
        temperature = float(input(f"Sensor {sensor_number} temperature: "))
        status = classify_temperature(temperature)

        if status == "INVALID":
            print("Invalid reading - skipped.")
            continue

        reading = (sensor_number, temperature, status)
        scan_readings.append(reading)

        if status == "CRITICAL":
            print("CRITICAL - scan stopped")
            warning_count += 1
            break
        elif status == "WARNING":
            print("WARNING")
            warning_count += 1
        else:
            print("SAFE")
            safe_count += 1

    return scan_readings, safe_count, warning_count

def show_report(reading_history):
    print("\n--- CURRENT REPORT ---")
    total_valid = len(reading_history)
    print(f"Valid readings: {total_valid}")
```

```
    if not reading_history:
        print("No valid sensor data yet.")
        return

    temperatures = [reading[1] for reading in reading_history]
    statuses = [reading[2] for reading in reading_history]

    print(f"Safe readings: {statuses.count('SAFE')}")
    print(f"Warnings: {statuses.count('WARNING')}")
    print(f"Critical readings: {statuses.count('CRITICAL')}")

    average = sum(temperatures) / len(temperatures)
    print(f"Average: {average:.1f} C")
    print(f"Highest: {max(temperatures):.1f} C")
    print(f"Lowest: {min(temperatures):.1f} C")

def format_reading(reading):
    sensor_number, temperature, status = reading
    return f"Sensor {sensor_number:<2} | {temperature:>6.1f} C | {status:<8}"

def show_history(reading_history):
    print("\n--- READING HISTORY ---")

    if not reading_history:
        print("No valid sensor readings recorded.")
        return

    lines = []
    for reading in reading_history:
        lines.append(format_reading(reading))

    print("\n".join(lines))

def show_recent(reading_history):
    print("\n--- RECENT READINGS ---")
    recent_readings = reading_history[-3:]

    if not recent_readings:
        print("No valid sensor readings recorded.")
        return

    for reading in recent_readings:
        print(format_reading(reading))

def show_status(reading_history, required_status):
    matching_readings = [
        reading for reading in reading_history
        if reading[2] == required_status
    ]

    print(f"\n--- {required_status} READINGS ---")

    if not matching_readings:
        print(f"No {required_status} readings recorded.")
        return

    for reading in matching_readings:
        print(format_reading(reading))

def show_highest_three(reading_history):
    print("\n--- HIGHEST THREE TEMPERATURES ---")
```

```
    if not reading_history:
        print("No valid sensor readings recorded.")
        return

    temperatures = [reading[1] for reading in reading_history]
    temperatures.sort()

    for temperature in temperatures[-3:][::-1]:
        print(f"{temperature:.1f} C")

def show_mission(mission_name, destination):
    print("\n--- MISSION INFORMATION ---")
    print(f"Mission: {mission_name}")
    print(f"Destination: {destination}")

    if len(mission_name) >= 3:
        print(f"Mission code: {mission_name[:3].upper()}")
    else:
        print("Mission code: " + mission_name.upper())

def search_history(reading_history):
    keyword = input("Search text: ").strip().lower()

    if keyword == "":
        print("Search text cannot be empty.")
        return

    matches = []

    for reading in reading_history:
        line = format_reading(reading)

        if keyword in line.lower():
            matches.append(line)

    print("\n--- SEARCH RESULTS ---")

    if not matches:
        print("No matching readings found.")
    else:
        print("\n".join(matches))

def status_command(reading_history):
    status = input(
        "Enter status (SAFE/WARNING/CRITICAL): "
    ).strip().upper()

    if status in ("SAFE", "WARNING", "CRITICAL"):
        show_status(reading_history, status)
    else:
        print("Invalid status.")

def show_log(reading_history):
    print("\n--- MISSION LOG ---")

    if not reading_history:
        print("No valid sensor readings recorded.")
        return

    lines = []
    for reading in reading_history:
```

```
        lines.append(format_reading(reading))

    log = "\n".join(lines)
    print(log)
    print(f"\nLog contains {len(lines)} readings.")

def show_help():
    command_text = (
        "scan report history recent safe warning critical "
        "top3 mission search status log help shutdown"
    )

    commands = command_text.split()

    print("\n--- AVAILABLE COMMANDS ---")
    print(" | ".join(commands))

def run_mission_control():
    reading_history = []

    mission_name = input("Mission name: ").strip().title()
    destination = input("Destination: ").strip().title()

    show_header(f"MISSION CONTROL v7.0 — {mission_name}")
    print(f"Destination: {destination}")

    while True:
        command = input(
            "\nCommand [scan/report/history/recent/safe/warning/"
            "critical/top3/mission/search/status/log/help/shutdown]: "
        ).strip().lower()

        if command == "shutdown":
            print("Shutting down Mission Control...")
            break

        if command == "report":
            show_report(reading_history)
            continue

        if command == "history":
            show_history(reading_history)
            continue

        if command == "recent":
            show_recent(reading_history)
            continue

        if command == "safe":
            show_status(reading_history, "SAFE")
            continue

        if command == "warning":
            show_status(reading_history, "WARNING")
            continue

        if command == "critical":
            show_status(reading_history, "CRITICAL")
            continue

        if command == "top3":
            show_highest_three(reading_history)
```

```
            continue

        if command == "mission":
            show_mission(mission_name, destination)
            continue

        if command == "search":
            search_history(reading_history)
            continue

        if command == "status":
            status_command(reading_history)
            continue

        if command == "log":
            show_log(reading_history)
            continue

        if command == "help":
            show_help()
            continue

        if command != "scan":
            print("Unknown command.")
            continue

        scan_readings, scan_safe, scan_warning = scan_sensors()
        reading_history.extend(scan_readings)

        print(
            f"Scan complete: {len(scan_readings)} valid readings, "
            f"{scan_safe} safe, {scan_warning} warning/critical."
        )

    print("=" * 60)
    print("SESSION ENDED".center(60))
    print("=" * 60)

run_mission_control()
```

Week 7 Concepts Used in v7.0

Concept	Where used
strip() / lower()	Command and search input
upper() / title()	Status and mission text
len()	Mission name, history and log counts
Indexing	Tuple fields and text positions
Slicing	Last three readings and mission code
in	Keyword search and status validation
split()	Help command string

join()	History, log and help display
f-strings	Headers, readings and reports
Alignment / precision	Formatted sensor readings and temperatures

Testing Checklist

- Run HELP and confirm all commands appear.
- Try commands with different capitalisation and surrounding spaces.
- Run HISTORY and RECENT before scanning.
- Run one scan, then test HISTORY, RECENT, STATUS and SEARCH.
- Search for safe, warning, critical and a value that does not exist.
- Try an empty search.
- Use STATUS with ' safe ', 'WARNING' and an invalid word.
- Use a mission name shorter than three characters to test the slice boundary.
- Run LOG before and after scans.
- Confirm the original v6.0 scan/report/status/top3 behaviour still works.
- Test a critical temperature and confirm the scan stops as in v6.0.

Reflection Questions

Question: Why is format_reading() useful?

Answer: It centralises the presentation of one sensor reading so several commands can reuse the same format.

Question: Why use '\n'.join(lines)?

Answer: It combines several strings into one multi-line string.

Question: What does mission_name[:3] do?

Answer: It extracts the first three characters.

Question: Why use keyword in line.lower()?

Answer: It provides a case-insensitive text search.

Question: Does v7.0 replace the list of tuples from v6.0?

Answer: No. The Week 6 data model remains the foundation.

Question: Which new Week 7 ideas are most visible?

Answer: String methods, indexing, slicing, membership, split(), join() and formatting.