MISSION CONTROL v5.0

WEEK 5 STUDENT WORKSHOP

FUNCTIONS: REFACTOR, REUSE, TEST

WHAT CHANGES THIS WEEK

This workshop changes the STRUCTURE of v4: we move meaningful jobs into functions, give those functions inputs and outputs, and keep the Week 3 decisions and Week 4 loops.

Dr Lei Wang (Griffith University)

What you are building

The final v5 should behave like the Week 4 continuous Mission Control system, but its code should be organised into small, meaningful functions.

- Week 1: arithmetic, expressions, types and formatting.
- Week 2: variables, assignment, input, conversion and output.
- Week 3: comparisons, Boolean logic and if/elif/else.
- Week 4: for, while, range, break, continue, pass, counters, totals and searching.
- Week 5: functions, parameters, arguments, return, None, default/keyword arguments and scope.

WEEK 5 DESIGN RULE

CLEAR JOB -> CLEAR NAME -> CLEAN INPUTS -> CLEAN OUTPUTS -> TEST -> REUSE

Mission 0 - Protect and run your v4

1. Make a copy of your working Week 4 file.
2. Keep the old file unchanged as your rollback copy.
3. Run v4 before making changes.
4. Try scan, report and shutdown.
5. Make sure you understand the counters, total, largest value, for loop, while loop, break and continue.

IMPORTANT

Do not refactor everything at once. In this workshop you will change ONE responsibility at a time, run the program, test it, and only then continue.

Full Week 4 reference code

Use this only to check your v4.

The code below is the complete continuous-monitor version from the previous workshop.

```
print("=" * 60)
print("MISSION CONTROL v4.0 - CONTINUOUS MONITOR".center(60))
print("=" * 60)

total_valid = 0
safe_count = 0
warning_count = 0
temperature_total = 0
largest_temperature = None

while True:
    command = input("\nCommand [scan/report/shutdown]: ").strip().lower()

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

    if command == "report":
        print("\n--- CURRENT REPORT ---")
        print(f"Valid readings: {total_valid}")
        print(f"Safe readings: {safe_count}")
        print(f"Warnings: {warning_count}")

        if total_valid > 0:
            average = temperature_total / total_valid
            print(f"Average: {average:.1f}")
            print(f"Highest: {largest_temperature:.1f}")
        else:
            print("No valid sensor data yet.")

        continue

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

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

        if temperature < 0:
            print("Invalid reading - skipped.")
            continue

        total_valid += 1
        temperature_total += temperature

        if largest_temperature is None or temperature > largest_temperature:
            largest_temperature = temperature

        if temperature >= 100:
            print("CRITICAL - scan stopped")
            warning_count += 1
            break
        elif temperature > 80:
            print("WARNING")
            warning_count += 1
        else:
            print("SAFE")
```

```
            safe_count += 1

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

The exact v4 structure is the starting point: a while command loop, a five-sensor for loop, continue for invalid readings, break for a critical reading, counters, total and largest-so-far state.

Mission 1 - Extract the first function

Start with one small responsibility: interpreting one temperature.

```
def classify_temperature(temperature, critical_limit=100):
    """Return the status for one valid temperature."""
    if temperature >= critical_limit:
        return "CRITICAL"
    elif temperature > 80:
        return "WARNING"
    else:
        return "SAFE"
```

Test it BEFORE integrating it:

```
print(classify_temperature(50))
print(classify_temperature(80))
print(classify_temperature(80.1))
print(classify_temperature(100))
print(classify_temperature(100, 90))
```

NOTICE

temperature is the parameter. 50, 80, 80.1 and 100 are arguments.

The function returns a value; it does not print the classification itself.

Mission 2 - Understand return

Compare print and return.

```
def version_a(x):
    print(x * 2)

def version_b(x):
    return x * 2

a = version_a(5)
b = version_b(5)

print("a =", a)
```

```
print("b =", b)
```

Predict before running:

- What is printed inside version_a?
- What is the value stored in a?
- What is the value stored in b?
- Why is return more useful when another part of the program needs the result?

Mission 3 - Average function and None

```
def calculate_average(total, count):
    """Return the average, or None when there is no valid data."""
    if count == 0:
        return None

    return total / count


print(calculate_average(300, 5))
print(calculate_average(120, 3))
print(calculate_average(0, 0))
```

Then use the result safely:

```
average = calculate_average(temperature_total, total_valid)

if average is None:
    print("No valid sensor data yet.")
else:
    print(f"Average: {average:.1f}")
```

KEY INSIGHT

The calculation function calculates.

The caller decides how to communicate the result to the user.

This separation makes the function reusable.

Mission 4 - Largest-so-far becomes a function

```
def update_largest(current_largest, value):
    """Return the largest value seen so far."""
    if current_largest is None or value > current_largest:
        return value

    return current_largest


largest = None
largest = update_largest(largest, 70)
```

```
largest = update_largest(largest, 55)
largest = update_largest(largest, 91)

print(largest)
```

Expected result: 91

This preserves the Week 4 largest-so-far algorithm, but now the algorithm has a name and can be reused.

The Week 4 workshop explicitly used this pattern with None and comparison.

Mission 5 - Default and keyword arguments

```
def classify_temperature(temperature, critical_limit=100):
    """Return the status for one valid temperature."""
    if temperature >= critical_limit:
        return "CRITICAL"
    elif temperature > 80:
        return "WARNING"
    else:
        return "SAFE"


print(classify_temperature(105))
print(classify_temperature(95, 90))
print(classify_temperature(95, critical_limit=90))
```

The second and third calls use the same critical limit.

The third makes the argument-to-parameter relationship explicit. This matches the Week 5 materials' treatment of default and keyword arguments.

Mission 6 - Refactor the sensor scan

Now move the Week 4 sensor loop into a function. The important design decision is that the function should RETURN the updated state to the caller.

```
def scan_sensors(total_valid, safe_count, warning_count,
                 temperature_total, largest_temperature):
    """Scan five sensors and return the updated mission state."""

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

        if temperature < 0:
            print("Invalid reading - skipped.")
            continue

        total_valid += 1
        temperature_total += temperature
        largest_temperature = update_largest(
            largest_temperature, temperature
```

```
        )

        status = classify_temperature(temperature)

        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 (total_valid, safe_count, warning_count,
            temperature_total, largest_temperature)
```

TRACE THE RESPONSIBILITIES

for -> repeat five sensors
continue -> skip invalid reading
classify_temperature() -> interpret one reading
break -> stop scan for critical reading
counters/total -> maintain state
update_largest() -> maintain largest value
return -> send updated state back

Mission 7 - Build the report function

```
def show_report(total_valid, safe_count, warning_count,
                temperature_total, largest_temperature):
    """Display the current Mission Control report."""

    print("\n--- CURRENT REPORT ---")
    print(f"Valid readings: {total_valid}")
    print(f"Safe readings: {safe_count}")
    print(f"Warnings: {warning_count}")

    average = calculate_average(temperature_total, total_valid)

    if average is None:
        print("No valid sensor data yet.")
    else:
        print(f"Average: {average:.1f}")
        print(f"Highest: {largest_temperature:.1f}")
```

Notice: show_report is intentionally a presentation function. It can print because displaying the report is its job.

Mission 8 - Build the command loop function

```
def run_mission_control():
    """Run the Mission Control command loop."""
```

```
    total_valid = 0
    safe_count = 0
    warning_count = 0
    temperature_total = 0
    largest_temperature = None

    while True:
        command = input(
            "\nCommand [scan/report/shutdown]: "
        ).strip().lower()

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

        if command == "report":
            show_report(
                total_valid,
                safe_count,
                warning_count,
                temperature_total,
                largest_temperature
            )
            continue

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

        (total_valid, safe_count, warning_count,
         temperature_total, largest_temperature) = scan_sensors(
            total_valid,
            safe_count,
            warning_count,
            temperature_total,
            largest_temperature
        )
```

This is the major Week 5 transformation: the command loop coordinates the system rather than containing every detail.

Mission 9 - FULL Mission Control v5

Here is the complete runnable v5.

You can copy this into a file named mission_control_v5.py and run it with Python 3.

```
def classify_temperature(temperature, critical_limit=100):
    """Return the status for one valid temperature."""
    if temperature >= critical_limit:
        return "CRITICAL"
    elif temperature > 80:
        return "WARNING"
    else:
        return "SAFE"


def calculate_average(total, count):
    """Return the average, or None when there is no valid data."""
    if count == 0:
        return None
```

```
    return total / count


def update_largest(current_largest, value):
    """Return the largest value seen so far."""
    if current_largest is None or value > current_largest:
        return value

    return current_largest


def show_report(total_valid, safe_count, warning_count,
                temperature_total, largest_temperature):
    """Display the current Mission Control report."""
    print("\n--- CURRENT REPORT ---")
    print(f"Valid readings: {total_valid}")
    print(f"Safe readings: {safe_count}")
    print(f"Warnings: {warning_count}")

    average = calculate_average(temperature_total, total_valid)

    if average is None:
        print("No valid sensor data yet.")
    else:
        print(f"Average: {average:.1f}")
        print(f"Highest: {largest_temperature:.1f}")


def scan_sensors(total_valid, safe_count, warning_count,
                 temperature_total, largest_temperature):
    """Scan five sensors and return the updated mission state."""
    for sensor_number in range(1, 6):
        temperature = float(
            input(f"Sensor {sensor_number} temperature: ")
        )

        if temperature < 0:
            print("Invalid reading - skipped.")
            continue

        total_valid += 1
        temperature_total += temperature
        largest_temperature = update_largest(
            largest_temperature, temperature
        )

        status = classify_temperature(temperature)

        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 (total_valid, safe_count, warning_count,
            temperature_total, largest_temperature)


def run_mission_control():
    """Run the Mission Control command loop."""
    total_valid = 0
```

```
    safe_count = 0
    warning_count = 0
    temperature_total = 0
    largest_temperature = None

    while True:
        command = input(
            "\nCommand [scan/report/shutdown]: "
        ).strip().lower()

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

        if command == "report":
            show_report(
                total_valid,
                safe_count,
                warning_count,
                temperature_total,
                largest_temperature
            )
            continue

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

        (total_valid, safe_count, warning_count,
         temperature_total, largest_temperature) = scan_sensors(
            total_valid,
            safe_count,
            warning_count,
            temperature_total,
            largest_temperature
        )


print("=" * 60)
print("MISSION CONTROL v5.0 - FUNCTIONS".center(60))
print("=" * 60)

run_mission_control()

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

DO NOT JUST COPY IT

First compare the structure with your v4. Then explain what each function does. Finally run v5 and test the same scenarios you tested in v4.

The purpose is to understand the refactoring, not merely to obtain a working program.

Function map - understand the architecture

Function	Job	Inputs	Output
classify_temperature	Interpret one reading	temperature, limit	status string
calculate_average	Calculate average	total, count	number or None

update_largest	Update largest-so-far	current, value	largest value
scan_sensors	Process five sensors	current state	updated state
show_report	Display report	summary state	None
run_mission_control	Coordinate commands	none	None

Why v5 is better than v4

v4	v5	Benefit
One large control block	Named functions	Each job is easier to find
Classification mixed into scan	classify_temperature()	Can test independently
Average calculation embedded in report	calculate_average()	Can reuse elsewhere
Largest logic embedded in scan	update_largest()	Algorithm has a clear name
Sensor loop mixed with command loop	scan_sensors()	Control flow is easier to read
State managed by the main function	Parameters + return	Dependencies are visible

This is the architectural change described in the Week 5 workshop: the behaviour remains, while meaningful pieces become named functions with clear inputs and outputs.

Mission 10 - Test v5 systematically

Run the COMPLETE v5 and record what happens.

Test	Input	Expected idea
Report before any scan	report	No valid sensor data yet.
All safe	scan: 50, 60, 70, 80, 75	Five valid readings; no warnings.
Boundary warning	scan: 80.1, 50, 60, 70, 75	80.1 is WARNING.
Invalid	scan: -5, 50, 60, 70, 75	-5 skipped.
Critical	scan: 50, 101, 60, 70, 80	Scan stops at 101.
Unknown command	hello	Unknown command.
Shutdown	shutdown	Program exits.

Mission 11 - Predict before you run

```
x = 5

def mystery(x, bonus=10):
    y = x + bonus
    return y * 2

result = mystery(x)

print(result)
print(x)
```

- What are the parameters?
- What is the argument?
- What is the default parameter?

- What is y?
- What is returned?
- What is printed?
- Does the global x change?

Expected output after you have reasoned it out: _____

Mission 12 - Scope detective

```
status = "READY"

def test():
    status = "BUSY"
    print("inside:", status)

print("before:", status)
test()
print("after:", status)
```

Predict all three lines. Then run it. Explain why the final global value remains unchanged.

Mission 13 - Use Python's tools

```
command = input("Command: ").strip().lower()

if command == "scan":
    print("Starting scan...")

help(str.strip)
help(str.lower)
help(round)
```

- strip() removes leading and trailing whitespace.
- lower() converts letters to lowercase.
- Both are methods called with dot notation.
- help() is a practical way to inspect Python functionality.

The Week 5 materials explicitly encourage students to use documentation and small experiments rather than memorising every function.

Mission 14 - Optional import challenge

```
import math

def required_sensor_cycles(readings, capacity):
    """Return the number of cycles needed."""
    return math.ceil(readings / capacity)

print(required_sensor_cycles(11, 5))
```

Do not memorise this function. Notice the workflow: import the module, use the module function, run a small test, and inspect the documentation when needed.

Debugging clinic

- If the function returns None unexpectedly: check whether every required path reaches return.
- If a value is unavailable outside the function: return it to the caller.
- If a function secretly uses a global variable: ask whether the value should be a parameter.
- If the scan does not stop: check the condition leading to break.
- If an invalid reading is counted: check that continue occurs before the counting code.
- If the average crashes: check the zero-count case.
- If a command is not recognised: check strip().lower() and the exact strings.
- If v5 behaves differently from v4: compare one function at a time rather than debugging the entire program at once.

Final student checklist

[ ] I can explain why v5 is a refactoring of v4, not a completely new program.

[ ] I can explain the job of every v5 function.

[ ] I can identify parameters and arguments.

[ ] I can explain return versus print.

[ ] I understand None.

[ ] I can use a default parameter and a keyword argument.

[ ] I can return multiple values and unpack them.

[ ] I understand local variables and eclipsing/shadowing.

[ ] I can explain why v5 uses a while loop and a for loop.

[ ] I can explain continue and break.

[ ] I can test boundary cases.

[ ] I can run the complete v5 code.

[ ] I can explain the code without reading every line.

THE WEEK 5 TAKEAWAY

Do not think of a function as just syntax.

Think: What meaningful job can I NAME, ISOLATE, TEST and REUSE?

Looking ahead

Keep both mission_control_v4.py and mission_control_v5.py. Next week, the new data-structure concepts will give these functions richer data to process. The goal is to keep the functions small, readable and reusable.