PROGRAMMING FUNDAMENTALS

WEEK 4 WORKSHOP

MISSION CONTROL v4

Dr Lei Wang (Griffith University)

THE SEMESTER STORY

Week 1 CALCULATE
Week 2 INTERACT
Week 3 DECIDE
Week 4 REPEAT
Week 5 ORGANISE & REUSE

Mission Control v3 could inspect a situation and make decisions.

In this workshop, you will make those decisions happen repeatedly.

You will use for, while, range(), break, continue and loop patterns to build a small monitoring system.

WORKSHOP RULE

Do not copy the final program first. Build one working piece at a time.
PREDICT -> CODE -> RUN -> TEST -> EXPLAIN

Mission Brief

Imagine Mission Control receives readings from several sensors. A real monitoring system cannot make one decision and stop. It should repeatedly process information, classify readings, record what happened, and stop for a meaningful reason.

YOUR MISSION

Upgrade your Week 3 Mission Control v3 into v4.

By the end, your system should be able to:
- check several sensors using a for loop;
- validate and classify readings;
- count safe and warning readings;
- calculate a running total and average;
- track the highest valid reading;
- use continue for invalid data;
- use break for an emergency condition;
- optionally run continuously until the operator shuts it down.

Learning Objectives

- Choose between for and while for a given task.
- Use range(start, stop, step) correctly.
- Combine loops with Week 3 Boolean expressions and if/elif/else.
- Use counters and accumulators.
- Use continue to skip invalid data.
- Use break to terminate a loop for an emergency or search condition.
- Understand pass as a placeholder.
- Trace loop state and test boundary cases.
- Build a program incrementally rather than jumping to the final version.

Before You Code: Get Your Week 3 File Ready

1. Find your working Mission Control v3 file.
2. Make a copy and rename it something like mission_control_v4.py.
3. Run v3 once. Confirm it still works before changing anything.
4. Keep v3.0 unchanged. It is your rollback point.

CHECKPOINT

Your v4 should be an evolution of your own program. If something breaks, compare it with v3 rather than starting again from nothing.

The Loop Design Checklist

- What must repeat?
- Do I know the number of repetitions/items in advance?
- What state must exist before the loop? (count, total, found, largest...)
- What decision happens inside each iteration?
- What changes during each iteration?
- What makes the loop stop?
- Do I need if inside the loop?

- Would break or continue make the intention clearer?
- What first, last and boundary cases should I test?

This checklist is the core problem-solving tool for this workshop. Use it before every mission.

Mission 1 - Visit Five Sensors

We know there are exactly five sensor positions. This is a definite repetition problem, so start with a for loop.

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

for sensor_number in range(1, 6):
    print(f"Checking sensor {sensor_number}...")
```

PREDICT BEFORE YOU RUN

What values does sensor_number take? How many times does the loop run?
Hint: range(1, 6) gives 1, 2, 3, 4, 5. The 6 is excluded.

Try these before running:

Change	Predict the values
range(5)	_____
range(2, 10, 2)	_____
range(5, 0, -1)	_____

CHECKPOINT

Remember: range(start, stop, step) includes start but excludes stop.

Mission 2 - Bring Week 3 Decisions Inside the Loop

Now each sensor provides a temperature. Reuse the Week 3 decision logic, but put it inside the loop so the same decision is made for every sensor.

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

    if temperature > 80:
        print("WARNING: temperature too high")
    else:
        print("Sensor within safe range")
```

PREDICT BEFORE YOU RUN

If the five readings are 70, 82, 90, 65, 81, how many warnings should appear?
Hint: Readings above 80 are warnings.

Upgrade it: add a safe counter and a warning counter.

```
safe_readings = 0
warning_readings = 0

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

    # Your Week 3 decision goes here.
    # Update exactly one counter for a valid reading.
```

CHECKPOINT
Initialise counters BEFORE the loop. If you put safe_readings = 0 inside the loop, the count keeps resetting.

Mission 3 - Reject Invalid Sensor Data with continue

A temperature below 0 is treated as invalid for this simplified exercise. Invalid data should not be classified as safe or warning data.

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

Insert this before the safe/warning decision.

PREDICT BEFORE YOU RUN
Why is continue useful here?
Hint: It ends only the current sensor's iteration. The next sensor can still be processed.

IMPORTANT
continue does NOT stop the for loop. It skips the remaining code for this one iteration and moves to the next sensor.

Test these readings:

Input	Expected treatment
-5	Invalid; skip
0	Valid; safe
80	Valid; safe
80.1	Valid; warning

Mission 4 - Track Valid Data

Now make the dashboard more informative. Track the number of valid readings and their total.

```
valid_readings = 0
temperature_total = 0

# Inside the loop, after invalid readings have been skipped:
valid_readings += 1
temperature_total += temperature
```

At the end, calculate the average only if there was at least one valid reading.

```
if valid_readings > 0:
    average_temperature = temperature_total / valid_readings
    print(f"Average temperature: {average_temperature:.1f}")
else:
    print("No valid readings received.")
```

PREDICT BEFORE YOU RUN
Why do we need the if valid_readings > 0 check?
Hint: It prevents division by zero when all readings are invalid.

CHECKPOINT
This is a good example of Week 3 logic protecting Week 4 computation.

Mission 5 - Track the Highest Reading

We also want the highest valid temperature. Do not use max(). Build the pattern yourself.

```
largest_temperature = None

# Inside the loop, after invalid readings have been skipped:
if largest_temperature is None or temperature > largest_temperature:
    largest_temperature = temperature
```

The first valid value becomes the initial candidate. Every later valid value is compared with the current largest.

PREDICT BEFORE YOU RUN
Why is None useful here? What should happen if there are no valid readings?
Hint: None represents 'we do not have a largest value yet'.

PATTERN TO REMEMBER
largest = None
for each valid value:
  if largest is None or value > largest:
    largest = value

Mission 6 - Emergency Shutdown with break

Suppose a temperature of 100 or higher is critical. Mission Control must stop scanning immediately rather than continue to later sensors.

```
if temperature >= 100:
    print("CRITICAL TEMPERATURE DETECTED")
    critical_found = True
    break
```

Add this after the reading has been validated, but before ordinary safe/warning classification.

PREDICT BEFORE YOU RUN
If the readings are 70, 85, 101, 60, 75, how many sensors are processed?
Hint: break exits the loop immediately when 101 is detected.

Keyword	In this mission
continue	Invalid reading: skip this sensor.
break	Critical reading: stop the entire scan.
pass	Reserved for a case where we intentionally need no action yet.

CHECKPOINT
Ask yourself: 'Do I want to skip this iteration, or leave the whole loop?' That tells you continue vs break.

Mission 7 - Build the Complete v4.0 Scan

Now assemble the pieces. Try it yourself first. Use the scaffold below rather than copying a finished solution.

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

safe_readings = 0
warning_readings = 0
valid_readings = 0
temperature_total = 0
critical_found = False
largest_temperature = None

for sensor_number in range(1, 6):

    temperature = float(input(f"Sensor {sensor_number} temperature: "))

    # 1. Invalid reading?
    #    If temperature < 0, print a message and continue.

    # 2. Critical reading?
    #    If temperature >= 100, set critical_found to True,
    #    print a message and break.

    # 3. Valid reading:
    #    update valid_readings
    #    update temperature_total

    # 4. Safe or warning?
    #    use Week 3 if/else
    #    update the appropriate counter
```

```
    # 5. Update largest_temperature

# 6. After the loop:
#    print average if valid_readings > 0
#    otherwise print no valid readings
#    print safe and warning counts
#    print whether scan was terminated or completed
#    print the highest valid reading
```

CHECKPOINT
Do not move on until your v4.0 works for ordinary readings, invalid readings and a critical reading.

v4.0 Reference Check

After you have attempted the build, compare your logic with the reference below. Focus on why each piece is where it is.

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

safe_readings = 0
warning_readings = 0
valid_readings = 0
temperature_total = 0
critical_found = False
largest_temperature = None

for sensor_number in range(1, 6):

    temperature = float(input(f"Sensor {sensor_number} temperature: "))

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

    if temperature >= 100:
        print("CRITICAL TEMPERATURE DETECTED")
        critical_found = True
        break

    valid_readings += 1
    temperature_total += temperature

    if temperature > 80:
        print("WARNING: temperature too high")
        warning_readings += 1
    else:
        print("Sensor within safe range")
        safe_readings += 1

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

if valid_readings > 0:
    average_temperature = temperature_total / valid_readings
    print(f"Average temperature: {average_temperature:.1f}")
else:
    print("No valid readings received.")
```

```
print(f"Safe readings: {safe_readings}")
print(f"Warnings: {warning_readings}")

if critical_found:
    print("Scan terminated for emergency response.")
else:
    print("Full scan completed.")

print(f"Highest reading: {largest_temperature}")
```

The supplied v4 code uses exactly these ideas: counters, a total, a largest-so-far value, a for/range loop, continue for invalid data and break for a critical condition.

Mission 8 - v4.1: Continuous Mission Control

Now we change the problem. We no longer know how many commands the operator will enter. This is where while becomes more natural.

NEW QUESTION
How do we keep Mission Control running until the operator explicitly chooses shutdown?

```
while True:
    command = input("Command [scan/report/shutdown]: ")

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

    if command == "report":
        print("CURRENT REPORT")
        # print the counters and summary
        continue

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

    # run one sensor scan here
```

The command loop is indefinite: we cannot predict how many commands the operator will issue. The word shutdown becomes the sentinel event.

CHECKPOINT
This is the key contrast: for handles a known collection of repetitions; while handles an ongoing process controlled by a condition/event.

v4.1 Complete Reference

After attempting the extension, compare your program with this version.

```
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]: ")

    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 supplied continuous-monitor version follows the same structure: while True for commands, break for shutdown, continue for report/unknown commands, and a for loop for the five-sensor scan.

Debugging Lab - Find the Loop Bugs

These are intentionally tricky. Predict what happens before running.

Bug A - The Loop Never Changes

```
i = 0

while i < 5:
    print(i)
    # i += 1 is missing
```

Question: Why does it never reach the stopping condition? Repair it.

Bug B - continue Creates an Infinite Loop

```
i = 0

while i < 5:
    if i == 2:
        continue
    print(i)
    i += 1
```

Question: What happens when i becomes 2? Which statement is skipped? Repair it without removing continue.

DEBUGGING HABIT

Whenever continue appears inside while, check whether it skips the statement that changes the loop condition.

Bug C - The Counter Resets

```
for n in range(1, 6):
    count = 0

    if n % 2 == 0:
        count += 1

print(count)
```

Question: Why is the final count wrong? Where should count = 0 be placed?

Mission Challenges

Challenge 1 - Count Multiples

Ask the user for a positive integer n. Use a single for loop to count how many numbers from 1 through n are divisible by 5. Do not use min(), max() or while.

```
# Your plan:
# n = ...
# count = ...
# for ...
#     if ...
#         ...
# print(...)
```

Challenge 2 - Average of Valid Readings

Repeatedly ask for temperatures until the user enters -999. Ignore readings below 0, and at the end print the average of valid readings. Use while and continue.

Challenge 3 - First Critical Sensor

Read up to 10 sensor values. If a value is at least 100, print the sensor number and stop immediately. Use for and break.

Challenge 4 - Operator Report

Extend v4.1 with a command called help that prints the available commands. The command loop should continue afterwards.

```
Command [scan/report/help/shutdown]:
```

Which statement should you use after displaying help? Why?

Quick Prediction Games

Work with a partner. Person A predicts. Person B checks. Then run the code.

Game 1

```
for i in range(2, 8, 2):
    print(i, end=" ")
```

Expected output: _____

Game 2

```
count = 0
for n in range(5):
    if n == 2:
        continue
    count += 1
print(count)
```

Expected output: _____

Game 3

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

Number of lines: _____

Game 4

```
x = 0
while x < 5:
    x += 2
print(x)
```

Final x: _____

CHECKPOINT
Do not run first. The prediction is the learning activity.

Test Your Mission Control Like a Programmer

Do not test only the 'normal' case.

Test	Example	What should happen?
All safe	50, 60, 70, 75, 80	All valid; no warnings.
Boundary	80 and 80.1	80 is safe; 80.1 is warning.
Invalid	-1	Skipped; not counted.
Critical	100	Critical; scan stops.
Critical early	100, 20, 30, 40, 50	Only the first sensor is processed.
No valid readings	-1, -2, -3, -4, -5	No average; no valid readings.
Highest first	90, 50, 60	Highest remains 90.
Highest last	50, 60, 90	Highest becomes 90.

BOUNDARY RULE
Test just below, exactly at, and just above important thresholds.

For 80, test 79.9, 80 and 80.1.

For 100, test 99.9, 100 and 100.1.

Explain Your Program Without Reading It

Pair activity: close your editor and explain the following.

Explain: Why is safe_readings initialised before the for loop?

Your explanation: _____

Explain: Why does invalid data use continue?

Your explanation: _____

Explain: Why does critical data use break?

Your explanation: _____

Explain: Why is valid_readings incremented only after invalid data is skipped?

Your explanation: _____

Explain: Why is temperature_total updated inside the loop?

Your explanation: _____

Explain: Why can the average only be calculated when valid_readings > 0?

Your explanation: _____

Explain: Why is largest_temperature initially None?

Your explanation: _____

Explain: Why is the outer while loop in v4.1 needed?

Your explanation: _____

Explain: Why does the command 'shutdown' use break?

Your explanation: _____

Explain: Why does the command 'report' use continue?

Your explanation: _____

What You Have Integrated

Week	Mission Control capability
Week 1	Arithmetic, expressions, types and formatting.
Week 2	Variables, assignment, input, conversion and output.
Week 3	Comparisons, Boolean logic and conditional decisions.
Week 4	for, while, range, break, continue, pass, counters, totals, filtering and searching.

THE BIG UPGRADE

v3: receive data -> make a decision -> report

v4: receive repeated data -> validate -> decide -> count/accumulate -> detect special cases -> report

v4.1: keep the system running -> accept commands -> scan -> report -> continue -> shutdown

Student Completion Checklist

- [ ] I created a separate mission_control_v4.py from my working v3.
- [ ] I can explain why the sensor scan uses for.
- [ ] I can explain range(1, 6) and the excluded stop value.
- [ ] I put Week 3 if/else logic inside the loop.
- [ ] I used a counter correctly.
- [ ] I used an accumulator correctly.
- [ ] I used continue for invalid readings.
- [ ] I used break for a critical reading.
- [ ] I understand why break and continue are different.
- [ ] I can explain pass.
- [ ] I tracked the largest value without max().
- [ ] I handled the no-valid-readings case.
- [ ] I tested boundary values.

- [ ] I traced at least one loop manually.
- [ ] I can explain why while is appropriate for the continuous command system.
- [ ] I completed at least one challenge without looking at the reference.
- [ ] I can explain every important line of my final program.

Key Takeaways

Remember	Why it matters
for = visit known values	Excellent for a fixed number of sensors or a sequence.
while = repeat while a condition/event allows it	Excellent for validation and ongoing interaction.
range stop is excluded	One of the most common loop mistakes.
break = leave the loop	Use for emergency stops and early search termination.
continue = next iteration	Use when the current item should be skipped.
pass = do nothing	Placeholder, not a loop-control shortcut.
count += 1	Counts qualifying events.
total += value	Builds a running result.
largest-so-far	Keeps the best value seen so far.
Initialise before the loop	Prevents counters/totals from resetting.
Trace state	Makes loop behaviour understandable.
Test boundaries	Finds many logical bugs.

THE DEEPEST IDEA

A loop is a controlled process in which program state changes over time.

If you can explain what changes, why it changes, and what makes the loop stop, you understand the algorithm.

Looking Ahead to Week 5 - Functions

Notice how the Mission Control program is getting longer. Some pieces of logic now have a clear job: checking a temperature, producing a report, or performing a scan. Next week, functions will let us package reusable logic into named units.

```
# Week 4: repeated logic inside a loop

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

    if temperature > 80:
        print("WARNING")
```

```
    else:
        print("SAFE")
```

Next week, this kind of logic can become a reusable function:

```
def check_temperature(temperature):
    if temperature > 80:
        return "WARNING"
    return "SAFE"
```

CHECKPOINT
Keep your working v4.0 and v4.1 files. They become the starting point for the next Mission Control upgrade.

Final Mission Debrief

What was the biggest difference between for and while?

Your answer: _____

Which part of v4 was hardest to build?

Your answer: _____

What did continue allow Mission Control to do?

Your answer: _____

What did break allow Mission Control to do?

Your answer: _____

What is one infinite-loop bug you can now diagnose?

Your answer: _____

Which boundary test was most useful?

Your answer: _____

How does v4 show the difference between 'process once' and 'process repeatedly'?

Your answer: _____

What part of Mission Control would you like to turn into a function next week?

Your answer: _____