PROGRAMMING FUNDAMENTALS

WEEK 3 WORKSHOP

MISSION CONTROL PANEL v3.0

Lei Wang (Griffith University)

THIS WEEK'S UPGRADE
v1.0 could CALCULATE.
v2.0 could INTERACT.
v3.0 can DECIDE.

DATA -> QUESTION -> TRUE/FALSE -> DECISION -> ACTION

This is a hands-on workshop.

You will upgrade the Mission Control program you built in Weeks 1 and 2.

Do not start by copying the finished program. Build it in stages.

WORKSHOP RULE
At every stage: PREDICT -> CODE -> RUN -> TEST -> EXPLAIN.
If your prediction is wrong, that is useful evidence. Investigate it.

Mission Brief

Last week Mission Control could collect mission information, store it in variables, perform calculations, and produce a report. But it treated every mission in essentially the same way.

This week we give the dashboard a new capability: judgement. It will convert mission rules into Boolean questions and use those answers to decide whether a mission should launch.

Version	Main capability	What Mission Control can do
v1.0	Calculate	Work with values, expressions and arithmetic.
v2.0	Interact	Ask for data, store it, process it and report it.
v3.0	Decide	Evaluate safety rules and choose actions.
Next	Repeat	Process many checks using loops.

DESIGN PRINCIPLE
Each week's new Python idea should solve a limitation of the previous version.
Ask: What can my program not do yet? What new concept could make it possible?

Learning Objectives

- Create and inspect True and False values.
- Use ==, !=, <, <=, > and >=.
- Distinguish assignment = from equality comparison ==.
- Store Boolean results in meaningful variables.
- Combine conditions with and, or and not.
- Use if, if/else and if/elif/else appropriately.
- Use correct indentation for conditional blocks.
- Use chained comparisons for ranges.
- Use f-strings to display changing information.
- Distinguish independent decisions from exclusive choices.
- Test normal cases and boundary cases.
- Debug a program by checking types, conditions, indentation and ordering.
- Connect Week 1 and Week 2 skills to the new Week 3 decision-making skills.

Before Coding: Recall Weeks 1 and 2

Mission Control still depends on everything you learned earlier.

Week 1 skill	Week 2 skill	Week 3 use

Numbers and arithmetic	input() and conversion	Compare user data with safety limits.
print()	Variables	Display Boolean diagnostics.
Expressions	f-strings / output	Explain decisions clearly.
// and %	INPUT -> PROCESS -> OUTPUT	The same program structure now includes decisions.

THINK BEFORE YOU RUN
What was the main limitation of Mission Control v2.0?
Hint: It accepted data and produced a report, but it did not change its behaviour according to the data.

Stage 1 - Your First Boolean Diagnostic

Start with one safety rule: fuel must be at least 70 percent.

```
fuel = float(input("Fuel level (%): "))

fuel_ok = fuel >= 70

print(f"Fuel level: {fuel}%")
print(f"Fuel OK: {fuel_ok}")
print(type(fuel_ok))
```

Before running, complete the prediction table.

Fuel	fuel >= 70	Expected type
69	_____	_____
70	_____	_____
71	_____	_____

THINK BEFORE YOU RUN
Why is fuel_ok a Boolean rather than a number?
Hint: The comparison asks a yes/no question.

KEY INSIGHT
A comparison is an expression.

Its result is a value, and that value has type bool.

Stage 2 - Build the Comparison Toolkit

Translate each mission question into Python.

Mission question	Python
Is fuel at least 70 percent?	fuel >= 70
Is wind at most 40 km/h?	wind <= 40
Is temperature above 80?	temperature > 80
Is status exactly online?	status == "online"
Is destination not restricted?	destination != "restricted"

The classic = versus == trap

```
x = 20       # assignment
x == 20      # comparison
```

MEMORY AID
= means ASSIGN. == means COMPARE.

THINK BEFORE YOU RUN
Which one changes x: x = 20 or x == 20? Which one asks a question?
Hint: One stores a value; the other produces True or False.

Stage 3 - Add Multiple Safety Questions

```
fuel = float(input("Fuel level (%): "))
wind = float(input("Wind speed (km/h): "))
temperature = float(input("Engine temperature: "))

fuel_ok = fuel >= 70
wind_ok = wind <= 40
temperature_ok = 10 <= temperature <= 80

print(f"Fuel OK:        {fuel_ok}")
print(f"Wind OK:        {wind_ok}")
print(f"Temperature OK: {temperature_ok}")
```

The temperature condition is a chained comparison. Read it as: temperature is between 10 and 80 inclusive.

THINK BEFORE YOU RUN
Why is it useful to create fuel_ok, wind_ok and temperature_ok instead of putting everything into one

enormous expression?
Hint: Named Boolean variables make the logic easier to read, test and debug.

PROGRAMMER HABIT
For a complicated decision, first create small Boolean questions. Then combine those answers.

Stage 4 - Combine Conditions with and

Launch requires all three safety checks to pass.

```
launch_ready = fuel_ok and wind_ok and temperature_ok
print(f"Launch ready: {launch_ready}")
```

fuel_ok	wind_ok	temperature_ok	launch_ready
True	True	True	_____
True	False	True	_____
False	True	True	_____
False	False	False	_____

MEMORY AID
and = ALL

Stage 5 - Add or and not

A realistic system can have alternative ways to satisfy a requirement.

```
primary_link = input("Primary link online? (yes/no): ") == "yes"
backup_link = input("Backup link online? (yes/no): ") == "yes"

communication_ok = primary_link or backup_link
print(f"Communication OK: {communication_ok}")
```

Now add a rule saying that an emergency lock must NOT be active.

```
emergency_lock = input("Emergency lock active? (yes/no): ") == "yes"

lock_clear = not emergency_lock
print(f"Lock clear: {lock_clear}")
```

Operator	Memory aid	Meaning
and	ALL	Every required condition must be True.
or	ANY	At least one condition must be True.
not	REVERSE	True becomes False; False becomes True.

THINK BEFORE YOU RUN
If the primary link is offline but the backup link is online, should communication_ok be True or False?
Hint: The rule uses or.

Stage 6 - Make Mission Control Act

Now turn the Boolean result into behaviour.

```
if launch_ready:
    print("LAUNCH AUTHORISED")
```

Notice the colon and indentation. The indented statement belongs to the if.

THINK BEFORE YOU RUN
What happens when launch_ready is False?
Hint: The indented block is skipped.

Stage 7 - Add else

```
if launch_ready:
    print("LAUNCH AUTHORISED")
else:
    print("LAUNCH HOLD")
```

KEY IDEA
if/else gives the program two paths. Exactly one branch executes.

Stage 8 - Make the Dashboard Helpful

A useful control panel should not simply say HOLD. It should explain what failed.

```
if not fuel_ok:
    print("Reason: fuel below required level")
```

```
if not wind_ok:
    print("Reason: wind exceeds safe limit")

if not temperature_ok:
    print("Reason: engine temperature outside safe range")

if not communication_ok:
    print("Reason: no communication link available")

if not lock_clear:
    print("Reason: emergency lock is active")
```

IMPORTANT DESIGN INSIGHT

These are separate if statements because several warnings may be true at the same time.

An if/elif/else chain would select only the first matching branch.

THINK BEFORE YOU RUN

Suppose fuel is low AND wind is too high. How many warning messages should appear?
Hint: More than one problem exists, so more than one warning should be possible.

Stage 9 - Multi-Way Decisions with elif

Mission Control can also classify a risk score.

```
risk_score = float(input("Mission risk score (0-100): "))

if risk_score >= 80:
    risk_level = "CRITICAL"
elif risk_score >= 60:
    risk_level = "HIGH"
elif risk_score >= 30:
    risk_level = "MODERATE"
else:
    risk_level = "LOW"

print(f"Risk level: {risk_level}")
```

THINK BEFORE YOU RUN

What risk level should 85 receive? What about 60, 30 and 29?
Hint: Remember: Python checks the chain from top to bottom.

FIRST MATCH WINS

In an if/elif/else chain, Python executes the first condition that is True and skips the remaining branches.

Stage 10 - Why Ordering Matters

```
# BAD ORDER
if score >= 50:
    grade = "PASS"
elif score >= 85:
    grade = "HIGH"

# A score of 90 reaches the first branch.
```

The second condition can never receive a score of 85 or higher because those values already satisfy score >= 50.

```
# BETTER ORDER
if score >= 85:
    grade = "HIGH"
elif score >= 50:
    grade = "PASS"
else:
    grade = "FAIL"
```

RULE
For descending thresholds, test the highest threshold first.

Stage 11 - Build Mission Control v3.0

Now combine the pieces. Do not paste the entire program at once. Build it in sections, running after each section.

```
# ============================================================
# MISSION CONTROL v3.0 - LAUNCH DECISION ENGINE
# ============================================================

print("=" * 66)
print("MISSION CONTROL v3.0 - LAUNCH DECISION ENGINE".center(66))
print("=" * 66)

# -------------------- INPUT --------------------
commander = input("Commander: ")
mission = input("Mission name: ")
destination = input("Destination: ")

fuel = float(input("Fuel level (%): "))
wind = float(input("Wind speed (km/h): "))
temperature = float(input("Engine temperature: "))

primary_link = input("Primary link online? (yes/no): ") == "yes"
backup_link = input("Backup link online? (yes/no): ") == "yes"
emergency_lock = input("Emergency lock active? (yes/no): ") == "yes"

# ---------------- BOOLEAN QUESTIONS ----------------
fuel_ok = fuel >= 70
```

```
wind_ok = wind <= 40
temperature_ok = 10 <= temperature <= 80
communication_ok = primary_link or backup_link
lock_clear = not emergency_lock

launch_ready = (
    fuel_ok
    and wind_ok
    and temperature_ok
    and communication_ok
    and lock_clear
)

# ---------------- DIAGNOSTICS ----------------
print("\n" + "-" * 66)
print("SAFETY DIAGNOSTICS".center(66))
print("-" * 66)

print(f"Fuel OK:              {fuel_ok}")
print(f"Wind OK:              {wind_ok}")
print(f"Temperature OK:       {temperature_ok}")
print(f"Communication OK:     {communication_ok}")
print(f"Emergency lock clear: {lock_clear}")

print("-" * 66)

# ---------------- DECISION ----------------
if launch_ready:
    print(f"LAUNCH AUTHORISED - Commander {commander}")
    print(f"Mission {mission} cleared for {destination}.")
else:
    print("LAUNCH HOLD")
    print("Safety review:")

    if not fuel_ok:
        print(" - Fuel below 70%")

    if not wind_ok:
        print(" - Wind exceeds 40 km/h")

    if not temperature_ok:
        print(" - Temperature must be between 10 and 80")

    if not communication_ok:
        print(" - Primary and backup communication links are offline")

    if not lock_clear:
        print(" - Emergency lock is active")

print("=" * 66)
```

BUILD, DO NOT COPY

If you can explain every section, you are learning.

If you only paste the finished program, you are missing the main purpose of the workshop.

Stage 12 - Test Like a Programmer

A program that works once is not necessarily correct. Test deliberately.

Test	Fuel	Wind	Temp	Primary	Backup	Lock	Expected
All safe	70	40	80	yes	no	no	LAUNCH
Fuel boundary fail	69	40	80	yes	no	no	HOLD
Wind boundary fail	70	41	80	yes	no	no	HOLD
Low temp boundary	70	40	9	yes	no	no	HOLD
Backup saves comms	75	30	50	no	yes	no	LAUNCH
Both links down	75	30	50	no	no	no	HOLD
Emergency lock	90	20	50	yes	yes	yes	HOLD

BOUNDARY TESTING

If the rule says fuel >= 70, test 69, 70 and 71.
If the rule says wind <= 40, test 39, 40 and 41.
If the rule says 10 <= temperature <= 80, test 9, 10, 80 and 81.

Connect Mission Control to Real Programming Problems

The same decision patterns appear in the kinds of programming problems you are solving in class and in assessment.

Grade Classification

```
mark = float(input("How many marks? "))

if mark >= 85:
    grade = 7
elif mark >= 75:
    grade = 6
elif mark >= 65:
    grade = 5
elif mark >= 50:
```

```
    grade = 4
else:
    grade = "F"

print(f"Grade awarded: {grade}")
```

TRANSFERABLE PATTERN
Threshold classification = if/elif/else + careful ordering + boundary testing.

Distance Bands and Discounts

```
distance = float(input("Distance (km): "))

if distance < 250:
    discount = 0
elif distance < 500:
    discount = 0.10
elif distance < 1000:
    discount = 0.15
elif distance < 2000:
    discount = 0.20
elif distance < 3000:
    discount = 0.35
else:
    discount = 0.50

print(f"Discount: {discount}")
```

Notice that after earlier cases have failed, later conditions can often be simpler. For example, elif distance < 500 already implies distance >= 250.

Ordering Three Values Without sort(), min() or max()

```
a = int(input("Integer 1: "))
b = int(input("Integer 2: "))
c = int(input("Integer 3: "))

if a < b:
    a, b = b, a

if a < c:
    a, c = c, a

if b < c:
    b, c = c, b

print("Descending:", a, b, c)
```

WHY THREE SEPARATE if STATEMENTS?
Each comparison may need to make a correction. We do not want to stop after the first correction.

Independent Decisions: Overtime + Bonus

```
hours = int(input("Hours worked: "))
cars = int(input("Cars sold: "))

rate = 36.25

if hours > 37:
    normal_pay = 37 * rate
    overtime_pay = (hours - 37) * rate * 1.5
else:
    normal_pay = hours * rate
    overtime_pay = 0

if cars > 5:
    bonus = (cars - 5) * 200
else:
    bonus = 0

salary = normal_pay + overtime_pay + bonus
print(f"Salary: ${salary:.2f}")
```

DECISION QUESTION

Ask: Are these choices exclusive, or can several decisions apply?
Overtime and bonus are independent, so both sets of conditions must be evaluated.

Debug Mission Control

Find the problems before running this code.

```
fuel = input("Fuel: ")

if fuel = 70
print("Fuel exactly 70")

if fuel >= 70:
    print("Safe")
elif fuel >= 90:
    print("Excellent")
```

Write down every problem you can find:

- What type does input() return?
- Is = correct here, or should it be ==?
- Where is the missing colon?
- Which line needs indentation?
- Is the if/elif threshold order sensible?

DEBUGGING CHECKLIST

Check: input type -> operator -> colon -> indentation -> condition order -> boundary cases.

Mission Upgrade Challenges

Choose at least TWO. Strong students should attempt more.

Challenge A - Oxygen safety

Add oxygen_level and require oxygen_level >= 90 before launch.

Your design / notes:

Challenge B - Crew readiness

Ask whether the crew is ready (yes/no) and include the result in launch_ready.

Your design / notes:

Challenge C - Weather override

Allow launch when wind is safe OR an approved weather override is active.

Your design / notes:

Challenge D - Risk classification

Add a risk score and classify it as CRITICAL, HIGH, MODERATE or LOW.

Your design / notes:

Challenge E - Fuel warning

If fuel is between 70 and 79 inclusive, print a warning but still allow launch.

Your design / notes:

Challenge F - Better diagnostics

Use f-strings to report every Boolean safety check clearly.

Your design / notes:

Challenge G - Boundary lab

Create at least three tests for every new numerical rule.

Your design / notes:

Optional Enrichment - Truthy and Falsy

Python can interpret some values as True or False in a Boolean context. Explore this after the main workshop.

```
print(bool(0))
print(bool(1))
print(bool(""))
print(bool("ready"))
print(bool("False"))
```

THINK BEFORE YOU RUN
Why is bool("False") True?
Hint: The string is non-empty. Python is not checking the meaning of the word.

BEGINNER BEST PRACTICE
For important Week 3 decisions, explicit comparisons are often clearer than relying on truthiness.

Optional Enrichment - Conditional Expressions

```
status = "LAUNCH" if launch_ready else "HOLD"
print(status)
```

This is useful for one short choice. For larger actions, normal if/else is usually clearer.

Student Checkpoint - Can You Explain These?

1. What is the type of True?

Your answer: _____

2. What does a comparison expression produce?

Your answer: _____

3. What is the difference between = and ==?

Your answer: _____

4. When is A and B True?

Your answer: _____

5. When is A or B True?

Your answer: _____

6. What does not do?

Your answer: _____

7. Why does indentation matter after if?

Your answer: _____

8. What is the difference between separate if statements and an if/elif/else chain?

Your answer: _____

9. Why should descending thresholds usually be ordered from highest to lowest?

Your answer: _____

10. What values would you test around x >= 50?

Your answer: _____

11. What does 10 <= temperature <= 80 mean?

Your answer: _____

12. Which Week 1 and Week 2 ideas are still inside Mission Control v3.0?

Your answer: _____

Quick Workshop Games

Game A - Left or Right: True or False?

Your instructor will show an expression. Raise your LEFT hand for True and your RIGHT hand for False.

```
x = 10

x > 5
x == 10
x != 10
x < 5
x > 5 and x < 20
x < 5 or x == 10
not x > 5
```

Do not run it first. Predict.

Game B - Which Branch?

```
score = 75

if score >= 85:
    print("HD")
elif score >= 75:
    print("D")
elif score >= 50:
    print("P")
else:
    print("F")
```

Raise your LEFT hand for HD/D/P and RIGHT hand for F. Then explain why.

Game C - Spot the Design Error

```
if score >= 50:
    grade = "PASS"
elif score >= 85:
    grade = "HD"
```

What is wrong? Can you repair it without changing the grading rules?

Final Mission Debrief

WHAT CHANGED THIS WEEK?

v1.0 - Mission Control could CALCULATE.
v2.0 - Mission Control could INTERACT.
v3.0 - Mission Control can INTERPRET data and DECIDE.

The important skill is not memorising if syntax. It is learning to turn a real rule into a Boolean question, combine questions correctly, select the right conditional structure, and test the boundaries.

Reflection

What new capability did v3.0 gain?

Your answer: _____

Which Boolean condition in your program is easiest to understand? Why?

Your answer: _____

Which condition was hardest to write?

Your answer: _____

What boundary test found a possible bug or surprised you?

Your answer: _____

When did you use separate if statements rather than elif?

Your answer: _____

What would you like Mission Control to monitor next week?

Your answer: _____

Week 3 Technical Checklist

- [ ] I can create and inspect True and False.
- [ ] I can use ==, !=, <, <=, > and >=.
- [ ] I can distinguish = from ==.
- [ ] I can store Boolean results in meaningful variables.
- [ ] I can combine conditions with and, or and not.
- [ ] I can write an if statement with correct indentation.
- [ ] I can write if/else.
- [ ] I can write if/elif/else.
- [ ] I understand first-match behaviour in an if/elif/else chain.
- [ ] I can order threshold conditions correctly.
- [ ] I can use chained comparisons.
- [ ] I can use f-strings to report results.

- [ ] I can distinguish independent decisions from exclusive choices.
- [ ] I can test just below, at, and just above a boundary.
- [ ] I can trace a conditional program before running it.
- [ ] I can explain every important part of Mission Control v3.0.

Key Takeaways

Remember	Why it matters
A comparison produces True or False.	It turns data into a Boolean question.
= assigns; == compares.	A very common syntax and exam mistake.
and = ALL.	Use when every requirement must pass.
or = ANY.	Use when alternative routes are acceptable.
not = REVERSE.	Use to express the opposite condition.
if = one-way choice.	Act only when a condition is true.
if/else = two-way choice.	Exactly one path is selected.
if/elif/else = categories.	First matching branch wins.
Separate if statements can all run.	Useful when several warnings/actions may apply.
Test boundaries.	Many logical bugs occur exactly at thresholds.
Build incrementally.	Small working stages are easier to understand and debug.

Looking Ahead to Week 4

Mission Control can now make one set of decisions. But a real monitoring system should not check one sensor and then stop. It should repeatedly process data.

NEXT UPGRADE
Week 4: REPEAT

Mission Control will learn to use while and for loops, range(), break, continue and pass. The decision logic you built today will become part of repeated processing.

Keep your working v3.0 file. It is the starting point for the next Mission Control upgrade.