PROGRAMMING FUNDAMENTALS

CHAPTER 3

BOOLEANS AND DECISIONS

Lei Wang (Griffith University)

THE BIG TRANSITION

Weeks 1 and 2 taught your programs to calculate and interact.

This chapter gives them judgement.

A program can now ask a yes/no question, interpret the answer, and choose what to do next.

Before Chapter 3	After Chapter 3
Calculate a result	Calculate and interpret a result
Display the same kind of output	Choose different output depending on data
Follow one fixed sequence	Select a path
Store information	Ask questions about information

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

What You Will Learn

By the end of this chapter, you should be able to build small programs that do more than calculate a value.

You will be able to translate rules into Boolean questions and use those questions to control program behaviour.

- Explain the bool type and the values True and False.
- Create Boolean values with comparison operators.
- Distinguish assignment = from comparison ==.
- Combine Boolean conditions using not, and and or.
- Use parentheses to make complicated logic clearer.
- Understand the logical precedence not > and > or.
- Explain short-circuit evaluation at an introductory level.
- Understand basic truthy and falsy behaviour and bool().
- Write one-way decisions with if.
- Write two-way decisions with if and else.
- Write multi-way decisions with if, elif and else.
- Use indentation correctly in conditional blocks.
- Use chained comparisons such as 10 <= temperature <= 80.
- Use f-strings to communicate decisions clearly.
- Recognise conditional expressions and match/case as useful Python extensions.
- Test boundary values and trace which branch executes.

The examples in this chapter deliberately connect concepts together.

The goal is not to memorise isolated syntax. The goal is to learn a repeatable way of designing decisions.

From Calculation to Behaviour

Imagine a Week 2 mission program. It asks for fuel and weather, then prints the values. That is useful, but it treats a dangerous mission exactly like a safe mission.

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

print("Fuel:", fuel)
print("Weather:", weather)
print("Mission configured.")
```

Now ask a different question: What should the program do if fuel is dangerously low? What if the weather is unsafe? This is the motivation for Boolean logic and selection.

THE KEY IDEA

Boolean expressions give Python the QUESTIONS.

Conditional statements give Python the ACTIONS.

Week	Core capability	Question the program can answer
1	Values, types, operators, expressions	What is the result?
2	Variables, strings, input, conversion, output	What data did the user provide, and what can I calculate?
3	Booleans and selection	What should I do because of that data?

A Useful Week 2 Upgrade: f-Strings

The first technique in this week's material is f-string formatting.

It is not the main decision-making concept, but it makes decision programs much easier to read.

```
product = "Apple"
price = 8.9

print(f"name = {product}, price = {price}")
print(f"{product} costs ${price}")
```

Put an f immediately before the opening quote. Anything inside { } is evaluated and inserted into the string.

```
name = "Sam"
score = 82

print(f"{name}'s score is {score}")
print(f"Next score target: {score + 5}")
```

USEFUL INSIGHT

The braces can contain expressions, not only variable names.

This means you can report a calculated value or a decision result directly.

THINK BEFORE YOU RUN

What will this print if fuel is 72?

print(f"Fuel OK: {fuel >= 70}")

Hint: The expression inside the braces is evaluated first.

The bool Type: Python's Yes/No Type

Python has a dedicated Boolean type, written bool. It has exactly two Boolean values:

```
True
False
```

Capitalisation matters. These are Python values, not ordinary strings.

```
print(True)
print(False)

print(type(True))
print(type(False))
```

MENTAL MODEL

Think of a Boolean as the answer to a yes/no question.

For example: Is the fuel sufficient? Is the password correct? Is the temperature safe?

Comparisons Create Boolean Values

Comparison operators ask questions about values. The answer is always True or False.

Operator	Question	Example
==	Are the values equal?	score == 50
!=	Are the values different?	status != 'offline'
>	Is the left value greater?	speed > 100
>=	Is it greater than or equal?	age >= 18
<	Is the left value smaller?	fuel < 20
<=	Is it smaller than or equal?	temperature <= 40

```
age = 20

print(age >= 18)
print(age < 18)
print(age == 20)
print(age != 20)
```

THINK BEFORE YOU RUN

What is the type of each result printed above?
Hint: A comparison produces a Boolean value.

The Most Important Trap: = versus ==

```
score = 80       # assignment
score == 80      # comparison
```

These symbols look similar but do completely different jobs.

Symbol	Meaning	Example	Question or action?
=	Assignment	score = 80	Store/change a value
==	Equality comparison	score == 80	Ask whether two values are equal

EXAM ALERT
A useful memory trick: one = puts a value somewhere; two == asks a question about equality.

Store Boolean Answers in Variables

```
fuel = 72
enough_fuel = fuel >= 30

print(enough_fuel)
print(type(enough_fuel))
```

Boolean values can be stored just like integers, floats and strings.

Good Boolean variable names often sound like yes/no questions.

```
is_ready = True
has_access = False
enough_fuel = fuel >= 30
```

NAMING INSIGHT
Names such as is_ready, has_access, systems_ok and enough_fuel make later conditions much easier to read.

Combining Questions: and, or, not

Real decisions often depend on more than one fact. Python provides three logical operators.

Operator	Meaning	When is it True?
and	Both conditions must hold	Only when both sides are True
or	At least one condition must hold	When one or both sides are True
not	Reverse a Boolean	True becomes False; False becomes True

```
fuel = 80
weather_clear = True

print(fuel >= 30 and weather_clear)
```

Both conditions must be true for the result of and to be True.

```
has_password = False
has_backup_code = True

print(has_password or has_backup_code)
```

At least one condition is true, so the result of or is True.

```
system_error = False
print(not system_error)
```

not reverses the Boolean value.

Truth Tables

A	B	A and B	A or B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

A	not A
False	True
True	False

HUMAN LOGIC FIRST

Before writing Python, say the rule in ordinary language.

For example: 'The mission can launch if fuel is sufficient AND weather is clear.' Then translate the word AND directly into Python's and.

The Order of Logical Operations

When a Boolean expression contains several logical operators, Python does not simply read them all from left to right. For logical operators, the priority is:

```
not   ->   and   ->   or
```

So not is evaluated before and, and and is evaluated before or.

```
x > 1 or x < 10 and not y < 0
```

Python interprets the logical structure as:

```
(x > 1) or ((x < 10) and (not (y < 0)))
```

BEST PRACTICE

You do not need to rely on memory when a condition becomes difficult to read.

Use parentheses to show your intended grouping explicitly.

THINK BEFORE YOU RUN

Which is clearer?

A) fuel >= 70 and wind <= 40 or weather == 'clear'
B) (fuel >= 70 and wind <= 40) or (weather == 'clear')

Hint: Both may be valid, but clarity matters.

Boolean Expressions: Build the Question in Pieces

A Boolean expression is any expression that evaluates to True or False.

Large Boolean expressions can be intimidating, so break them into smaller questions.

```
x = 20
y = 30
a = "John"
b = "David"

result = x >= 25 or y < 50 and not a > b
print(result)
```

Instead of mentally solving the whole line at once, decompose it:

```
question1 = x >= 25
question2 = y < 50
question3 = a > b
```

```
print(question1)
print(question2)
print(question3)
```

PROBLEM-SOLVING HABIT
For a complex condition: calculate the small comparisons first, label their True/False results, then combine those results with and/or/not.

Short-Circuit Evaluation

For “and” and “or”, Python evaluates from left to right and may stop as soon as the final result is already known.

```
False and print("Will this run?")
True or print("Will this run?")
```

In the first line, False and anything must be False, so Python does not need the right side.

In the second line, True or anything must be True, so Python does not need the right side.

WHY YOU SHOULD CARE
Short-circuiting is useful because a later condition can be checked only when an earlier condition makes that check relevant. This becomes increasingly useful in larger programs.

Truthy and Falsy Values

Python can interpret many values as True or False in a Boolean context. You can see the conversion explicitly with bool().

```
print(bool(0))
print(bool(1))
print(bool(-10))

print(bool(""))
print(bool("Python"))
print(bool("False"))
```

Value	Boolean interpretation
0	False
Non-zero number	True
"" (empty string)	False

Any non-empty string	True

SURPRISE
bool("False") is True. The string is not empty.

Python is checking whether the string contains something, not whether its characters spell the word False.

For beginners, explicit comparisons are often clearer. For example, systems_online == "yes" communicates the intended question more clearly than relying on truthiness.

if: Turn a Question into Action

An if statement allows a program to execute a block of code only when a condition is True.

```
fuel = 20

if fuel < 30:
    print("WARNING: Low fuel")
```

Read it as: IF fuel is below 30, THEN execute the indented statement.

CRITICAL IDEA
The condition after if must evaluate to True or False.

The indented block executes only when the condition is True.

Indentation Is Part of Python Syntax

```
fuel = 20

if fuel < 30:
    print("WARNING")
    print("Refuel before launch")

print("System check complete")
```

The first two print statements belong to the if block. The final print is outside the block and runs regardless.

THINK BEFORE YOU RUN
What would change if 'System check complete' were indented?
Hint: It would become part of the if block.

One-Way, Two-Way and Multi-Way Decisions

One-Way: if

```
temperature = 85

if temperature > 80:
    print("Temperature warning")
```

Use this when an action is needed only when a condition is True.

Two-Way: if / else

```
age = int(input("Age: "))

if age >= 18:
    print("Adult ticket")
else:
    print("Junior ticket")
```

Exactly one branch executes: either the if branch or the else branch.

Multi-Way: if / elif / else

```
score = float(input("Score: "))

if score >= 85:
    grade = "HD"
elif score >= 75:
    grade = "D"
elif score >= 65:
    grade = "C"
elif score >= 50:
    grade = "P"
else:
    grade = "F"

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

FIRST MATCH WINS

Python tests an if/elif/else chain from top to bottom and executes the first matching branch.

After a branch is selected, later branches in that chain are not tested for execution.

A Classic Ordering Bug

```
# WRONG FOR A DESCENDING THRESHOLD SYSTEM

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

A score of 90 satisfies score >= 50, so Python chooses P and never reaches the HD condition.

DEBUGGING RULE
When using descending thresholds, test the highest threshold first.

Boundary Thinking: Where Bugs Hide

Conditional programs often fail at boundaries: values exactly equal to a threshold.

Do not test only an ordinary value.

Condition	Useful boundary tests
age >= 18	17, 18, 19
score >= 50	49, 50, 51
fuel < 30	29, 30, 31
temperature <= 80	79, 80, 81

THE 3-VALUE RULE
For a threshold, test just below it, exactly on it, and just above it.

This simple habit catches many mistakes with < versus <= and > versus >=.

Chained Comparisons

Python allows comparisons to be written in a form that resembles mathematics.

```
x = 0.7
print(0 < x <= 1)
```

This is equivalent to:

```
x > 0 and x <= 1
```

It is particularly useful for range checks.

```
temperature = 25
safe = 15 <= temperature <= 35
print(safe)
```

THINK BEFORE YOU RUN

For temperature = 35, is 15 <= temperature <= 35 True or False?
Hint: Both endpoints are included.

Compact Choices: Conditional Expressions

For a very small choice, Python has a compact conditional expression.

```
x = 15
y = 1 if x > 10 else 0

print(y)
```

The same idea using a normal if/else is:

```
if x > 10:
    y = 1
else:
    y = 0
```

Conditional expressions are useful for short, simple choices. Do not sacrifice readability just to make a program shorter.

```
print(x if x > 0 else -x)
```

GOOD PRACTICE
Compact code is not automatically better code.

If the decision has several actions or complicated logic, use a normal if/elif/else structure.

Putting It Together: Mission Control Learns to Decide

Mission Control v1.0 could calculate.

Mission Control v2.0 could receive, store and process changing mission data.

Now v3.0 can interpret that data and make a launch decision.

MISSION CONTROL v3.0
DATA -> QUESTIONS -> BOOLEAN RESULTS -> COMBINED DECISION -> ACTION

```
print("=" * 64)
print("MISSION CONTROL v3.0 - LAUNCH DECISION ENGINE".center(64))
print("=" * 64)

commander = input("Commander: ")
mission = input("Mission name: ")
```

```
fuel = float(input("Fuel level (%): "))
wind_speed = float(input("Wind speed (km/h): "))
temperature = float(input("Engine temperature (C): "))
systems_online = input("All systems online? (yes/no): ")

fuel_ok = fuel >= 70
wind_ok = wind_speed <= 40
temperature_ok = 10 <= temperature <= 80
systems_ok = systems_online == "yes"

print("\n--- BOOLEAN DIAGNOSTICS ---")
print(f"Fuel OK:        {fuel_ok}")
print(f"Wind OK:        {wind_ok}")
print(f"Temperature OK: {temperature_ok}")
print(f"Systems OK:     {systems_ok}")

launch_ready = fuel_ok and wind_ok and temperature_ok and systems_ok

print(f"Launch ready:   {launch_ready}")
print("-" * 64)

if launch_ready:
    print(f"LAUNCH AUTHORISED - Good luck, Commander {commander}!")
else:
    print("LAUNCH HOLD - one or more safety checks failed.")

print("=" * 64)
```

Why This Example Matters

- input(), variables, conversion and output come from Weeks 1 and 2.
- Every safety rule becomes a Boolean question.
- Boolean variables make the final decision easier to read.
- and combines independent safety requirements.
- if turns the final Boolean result into different behaviour.
- f-strings communicate changing data clearly.

Test the Decision Boundaries

Safety rule	Test values
fuel >= 70	69, 70, 71
wind_speed <= 40	39, 40, 41
10 <= temperature <= 80	9, 10, 80, 81
systems_online == "yes"	yes, no

PROGRAMMER'S QUESTION

Do not ask only 'Does my program run?'

Ask 'Does it behave correctly for every important case I can think of?'

A Better Decision System Explains Why

A useful program often should not simply say NO. It should help the user understand the reason.

```
if not fuel_ok:
    print("Reason: insufficient fuel")

if not wind_ok:
    print("Reason: wind speed too high")

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

if not systems_ok:
    print("Reason: systems are not all online")
```

DESIGN INSIGHT

Good software is designed for humans as well as computers.

A decision that explains its reason is much more useful than a mysterious YES or NO.

Where Boolean Decisions Appear in Real Programs

Theme Park Ride Controller

```
height = int(input("Height (cm): "))
age = int(input("Age: "))

can_ride = height >= 140 and age >= 12

if can_ride:
    print("Ride access approved!")
else:
    print("Access denied for safety.")
```

The program turns a real-world rule into a Boolean expression. The same pattern can be used for eligibility, safety checks and access control.

Tiny Login Simulator

```
saved_user = "student"
saved_pin = "3141"

user = input("Username: ")
pin = input("PIN: ")

username_ok = user == saved_user
pin_ok = pin == saved_pin

if username_ok and pin_ok:
    print("ACCESS GRANTED")
else:
```

```
    print("ACCESS DENIED")
```

IMPORTANT
This is a teaching simulation, not secure authentication.

The programming lesson is equality comparison plus and.

From English Rules to Python

One of the most important programming skills is translating a human rule into a Boolean expression.

English rule	Python
Age is at least 18	age >= 18
Temperature is between 10 and 30 inclusive	10 <= temperature <= 30
User is admin AND account is active	is_admin and account_active
Fuel is low OR engine is overheating	fuel < 20 or temperature > 90
System is not offline	not system_offline

Practise in both directions: English -> Python and Python -> English.

THINK BEFORE YOU RUN
Translate: 'A student receives the discount if they are a student OR a senior, AND the promotion is active.'
Hint: Try writing the grouped Boolean expression before writing any if statement.

```
discount = (is_student or is_senior) and promotion_active
```

Optional Python Extension: match / case

Now we introduce match/case. It is useful, but if/elif/else should remain your main tool for threshold decisions at this stage.

```
grade = 7

match grade:
    case 7:
        print("marks >= 85")
    case 6:
        print("marks >= 75")
    case 5:
        print("marks >= 65")
```

```
    case _:
        print("marks < 65")
```

match evaluates an expression and selects a matching case. The underscore _ acts as the default case.

Multiple Alternatives

```
state = input("Enter state: ")

match state:
    case "Queensland" | "QLD":
        print("Queensland")
    case "Victoria" | "VIC":
        print("Victoria")
    case "NSW" | "New South Wales":
        print("New South Wales")
    case _:
        print("State not recognised")
```

For beginners, think of match/case as a useful extension for matching discrete values. Do not use it to replace every if/elif/else program.

Common Mistakes and How to Debug Them

Mistake	What is wrong?	Better approach
if x = 10:	= is assignment, not comparison.	Use if x == 10:
if x == 10	Missing colon.	Add : after the condition.
Unindented branch	Python uses indentation to define the block.	Indent the statements belonging to the branch.
Low threshold tested first	A broad condition may capture the value too early.	For descending thresholds, test high values first.
bool("False") expected False	Non-empty strings are truthy.	Use an explicit comparison if that is what you mean.
Very complex condition	Hard for humans to read/debug.	Break it into Boolean variables or add parentheses.
Separate if statements for exclusive choices	More than one action may execute.	Use if/elif/else when exactly one branch should be selected.

DEBUGGING MINDSET

A program can run without being correct. Syntax errors are only one type of problem. Logical errors are often found by tracing conditions and testing carefully chosen values.

Practice: Predict Before You Run

Boolean Prediction

```
x = 8

print(x > 5)
print(x == 8)
print(x != 8)
print(x > 5 and x < 10)
print(x < 5 or x == 8)
print(not x > 5)
```

1. Write True or False beside every line before running it.
2. Explain each answer using smaller questions.
3. Run the program and compare.

Branch Prediction

```
score = 75

if score >= 85:
    print("A")
elif score >= 75:
    print("B")
elif score >= 50:
    print("C")
else:
    print("D")
```

THINK BEFORE YOU RUN
How many branches execute? Why?
Hint: Only one branch in this chain is selected.

Challenge: Boolean Detective

Given:

```
x = 20
y = 5
```

Evaluate each expression manually:

```
x > 10 and y < 10
x < 10 or y == 5
not (x == 20)
x > 10 and not y > 10
0 < y <= 5
```

1. Write the result of each small comparison.

2. Combine the results.
3. Only then run the code to check yourself.

Challenge: Eligibility Checker

Design a scholarship eligibility program with these rules:

- Academic score must be at least 80.
- Attendance must be at least 90%.
- The application must be completed.
- All three conditions must be true for eligibility.

A strong solution begins with separate Boolean variables:

```
score_ok = score >= 80
attendance_ok = attendance >= 90
application_ok = application_completed

eligible = score_ok and attendance_ok and application_ok
```

Then decide what to print based on eligible.

The Decision Design Recipe

When you are given a programming problem involving rules or choices, use this process.

1. Identify the data you need.

2. Ask the yes/no questions implied by the problem.

3. Write each question as a Boolean expression.

4. Test each Boolean expression independently.

5. Combine them with and, or or not if necessary.

6. Decide what should happen when the result is True and when it is False.

7. Choose if, if/else, or if/elif/else.

8. Test normal cases and boundary cases.

WHY THIS WORKS
It separates the problem into two layers: first decide what is true, then decide what the program should do. This prevents many beginners from trying to write the entire if statement in one step.

Quiz and Exam Focus

For assessment questions, be particularly confident with:

- The value and type of Boolean expressions.
- ==, !=, >, >=, < and <=.
- = versus ==.
- The meaning of not, and and or.

- Logical precedence: not before and before or.
- Using parentheses to make logic clear.
- Basic truthy and falsy rules.
- Tracing which if/elif/else branch executes.
- The fact that an if/elif/else chain selects the first matching branch.
- Threshold ordering.
- Boundary values.
- Indentation and colons.
- Translating English decision rules into Python.

EXAM HABIT

When tracing a conditional program, write down the value of each variable first.

Then evaluate each condition as True or False. Only after that decide which branch executes.

Self-Test

1. What is the type of True?

Answer: _____

2. What values can a bool contain?

Answer: _____

3. What does a comparison expression produce?

Answer: _____

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

Answer: _____

5. When is A and B True?

Answer: _____

6. When is A or B True?

Answer: _____

7. What does not do?

Answer: _____

8. What is the logical precedence of not, and and or?

Answer: _____

9. Why can parentheses improve a Boolean expression?

Answer: _____

10. What does short-circuit evaluation mean?

Answer: _____

11. Why is bool('False') True?

Answer: _____

12. What does an if statement do?

Answer: _____

13. Why does indentation matter after if?

Answer: _____

14. What is the difference between if, if/else and if/elif/else?

Answer: _____

15. Why does condition order matter in an if/elif/else chain?

Answer: _____

16. What three values should you test around a threshold?

Answer: _____

17. What does 10 <= x <= 20 mean?

Answer: _____

18. When is a conditional expression useful?

Answer: _____

19. What is match/case useful for?

Answer: _____

20. How does Mission Control v3.0 differ from v2.0?

Answer: _____

Chapter Concept Map

Concept	Role in a program	Example
bool	Represents a yes/no result	True
comparison	Asks a question	fuel >= 70
and	Requires both conditions	fuel_ok and wind_ok
or	Allows alternatives	is_student or is_senior
not	Reverses a condition	not system_error
if	Acts when a condition is true	if fuel < 30:
else	Provides the alternative path	else:
elif	Adds another exclusive choice	elif score >= 75:
chained comparison	Checks a range	10 <= x <= 80
f-string	Communicates changing data	f"Fuel: {fuel}"

match/case	Matches discrete patterns/values	case "QLD":

Chapter Checklist

- [ ] I can explain True and False.
- [ ] I can use all six comparison operators.
- [ ] I can distinguish = from == without hesitation.
- [ ] I can combine Boolean conditions with and, or and not.
- [ ] I know the logical precedence not > and > or.
- [ ] I can use parentheses when a condition is complicated.
- [ ] I understand basic truthy/falsy behaviour.
- [ ] I can write an if statement with correct indentation.
- [ ] I can write if/else and if/elif/else decisions.
- [ ] I understand why branch ordering matters.
- [ ] I can test just below, at, and just above a threshold.
- [ ] I can write a chained comparison.
- [ ] I can use an f-string to report a result.
- [ ] I can explain short-circuit evaluation at a basic level.
- [ ] I can translate an English rule into a Boolean expression.
- [ ] I can explain Mission Control v3.0 in terms of data, questions, decisions and actions.

Summary

The most important idea in this chapter is not the spelling of if. It is the relationship between information, questions and behaviour.

A comparison turns data into a Boolean answer. Boolean operators combine those answers. An if statement uses the answer to select what the program does. This is the point where programming moves from simply calculating results to controlling behaviour.

The same fundamentals you learned in Weeks 1 and 2 are still underneath everything: values, types, variables, input, conversion, arithmetic, expressions and output. Week 3 adds the ability to interpret those results and choose a path.

THE THREE-WEEK STORY
Week 1: CALCULATE
Week 2: INTERACT
Week 3: DECIDE

Mission Control can now make decisions once. But real systems often need to make decisions repeatedly: checking sensors, processing many numbers, counting events, waiting for valid input, or repeating an operation.

NEXT CHAPTER: Chapter 4 will introduce ITERATION: while, for, range, break, continue and pass. The key question will change from 'What should I do?' to 'What should I keep doing, and when should I stop?'