Lab 02: Booleans & Logic

COMP 102 — Introduction to Computing
Forman Christian University — Spring 2026

A Walkthrough Tutorial
Duration: ∼3 hours — Based on Lectures 3 & 4

How to Use This Lab

This lab is a **guided walkthrough** — not a test. For every exercise, follow this pattern:

Step 1: Predict — Write down what you think will happen *before* touching the computer.

Step 2: Type — Enter the code in Thonny's **Shell** (interpreter) or **Code Editor**.

Step 3: Verify — Compare your prediction with the actual result. If they differ, figure out *why*.

Setup: Open **Thonny** on your computer. You should see the Shell window at the bottom (with `>>>`).

1 Review: Strings & I/O (∼15 min)

Ref: Lecture 3 — Strings and I/O slides

Shell vs Code Editor

Use the **Shell** (`>>>`) for short, one-line experiments. When the code is more than 3–4 lines (especially code using `input()`), type it in Thonny's **code editor** (top pane) and press **Run** (or **F5**).

Exercise 1 **Very Easy**

Predict the output of each `print` statement, then verify:

```
>>> x = "Best"
>>> y = 3
>>> z = "goals"
>>> print(x, y, z)
```

Your prediction: ______________________
*Hint: When `print` receives multiple arguments separated by commas, it puts **spaces** between them.*

Exercise 2 Very Easy

Predict the output, then verify:

```python
>>> x = "Best"
>>> y = 3
>>> z = "goals"
>>> print(x + str(y) + z)
```

Your prediction: ____________________
Why is `str(y)` needed here?

Exercise 3 Medium

Predict first! Type this in the **code editor** and run it. When prompted, enter `7`. What does each line print?

```python
w = input("Enter: ")
print(w * 2)
print(int(w) * 2)
```

Line 2 prints: ____________________ Line 3 prints: ____________________
Why are they different?

Exercise 4 Medium

Type this in the **code editor** and run it. Identify the error(s) and fix them:

```python
x = 42
y = 2.5
z = "age"
msg1 = "x is " + x
msg2 = "y is " + y
msg3 = "z is " + z
```

Which line(s) cause an error and why?

Exercise 5 Medium

Predict what each line prints, then verify:

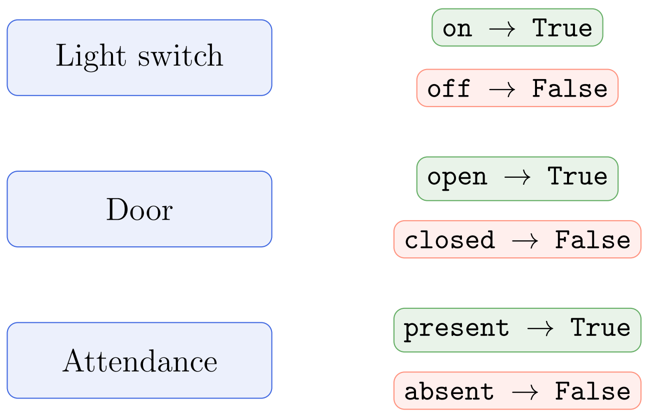
```python
dist = 3
print(f"Half distance: {dist/2} km")
print(f"Distance squared: {dist**2} km")
```

Line 2 prints: ____________________

Line 3 prints: ____________________

2 The Boolean Type (∼15 min)

Ref: Lecture 4 — The Boolean Type slides



Everything is either True or False — that's a Boolean!

Exercise 6 **Very Easy**

Type each line in the Shell. Predict the output of each **before** you type it.

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

Your predictions: ____________________ ____________________

____________________ ____________________

What do you notice about the capitalization of `True` and `False`?

Case Matters!

Python is **case-sensitive**. Only `True` and `False` (with a capital letter) are valid booleans. Writing `true`, `FALSE`, or `TRUE` will give you a `NameError`.

Exercise 7 **Medium**

Careful! Which of the following are **valid** in Python? Write **Valid** or **Error** next to each:

(a) `x = True` __________

(b) `x = true` __________

(c) `x = FALSE` __________

(d) `x = False` __________

(e) `x = "True"` ____________

For part (e): What is the **type** of `x`? Is it a `bool` or a `str`?

Exercise 8 Medium

Predict the output, then verify:

```
>>> a = True
>>> b = "True"
>>> print(type(a))
>>> print(type(b))
>>> print(a == b)
```

Your predictions:
`type(a)`: ________________________
`type(b)`: ________________________
`a == b`: ________________________

Exercise 9 Very Easy

Predict the output:

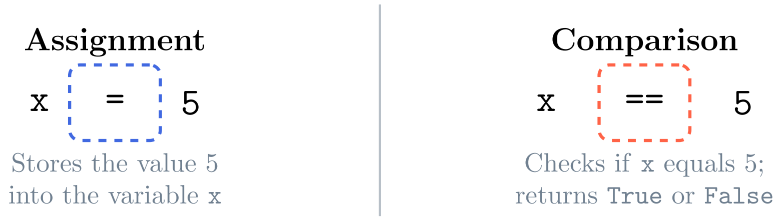
```
>>> exercise = True
>>> coding = False
>>> print(exercise)
>>> print(type(coding))
```

Your predictions: ________________________ ________________________
Booleans can be stored in variables, just like `int` or `str`.

3 Comparison Operators (∼25 min)

Ref: Lecture 4 — Comparison Operators slides

Assignment vs Comparison


Assignment
x = 5
Stores the value 5 into the variable x
Comparison
x == 5
Checks if x equals 5; returns True or False

The Six Comparison Operators

`i < j`	→	less than
`i <= j`	→	less than or equal to
`i > j`	→	greater than
`i >= j`	→	greater than or equal to
`i == j`	→	equality test (**not** `=`)
`i != j`	→	inequality test (not equal)

Each comparison returns either `True` or `False`.

Exercise 10 — Very Easy

Predict the result of each comparison, then verify:

```
>>> 8 == 8
>>> 8 == 3
>>> 8 != 3
>>> 8 != 8
```

Your predictions: ____________________ ____________________

____________________ ____________________

Exercise 11 — Very Easy

Predict the result, then verify:

```
>>> 4 < 9
>>> 9 < 4
>>> 9 > 4
```

Your predictions: ____________________ ____________________ ____________________

Exercise 12 — Very Easy

Watch out for `<=` and `>=`! Predict the result, then verify:

```
>>> 7 <= 7
>>> 7 >= 8
>>> 7 <= 6
```

Your predictions: ____________________ ____________________ ____________________

Why is `7 <= 7` `True`?

Exercise 13 — Very Easy

You can compare **expressions**, not just simple values. Predict the result, then verify:

```
>>> (3 * 4) == 12
>>> len("python") > 4
```

Your predictions: ____________ ____________

Each side is evaluated first, then compared.

Exercise 14 Medium

Types and comparisons. Predict the result, then verify:

```
>>> 5 == 5.0
>>> type(5) == type(5.0)
>>> "5" == 5
>>> len("hello") == 5
```

Your predictions: ____________ ____________

____________ ____________

Why is `5 == 5.0` `True` but `type(5) == type(5.0)` `False`?

Exercise 15 Medium

Now let `m = 12` and `n = 5`. Predict the result:

```
>>> m = 12
>>> n = 5
>>> m >= 12
>>> m == (n * 2)
>>> (m // n) == 2
>>> abs(n - m) > 5
```

Your predictions: ____________ ____________

____________ ____________

Exercise 16 Medium

Predict the value **and type** of each:

```
>>> 6 + 2
>>> 6 + 2 == 8
>>> type(6 + 2)
>>> type(6 + 2 == 8)
```

Your predictions: ____________ ____________

____________ ____________

Notice how adding `== 8` changes the type from `int` to `bool`.

Exercise 17 Hard

Tricky! Predict the result, then verify:

```
>>> 1.1 + 2.2 == 3.3
>>> 1.1 + 2.2
```

Your predictions: ____________________ ____________________

Why does the first line return an unexpected result?

4 Logical Operators: `not`, `and`, `or` (∼25 min)

Ref: Lecture 4 — Logical Operators slides

`not`

A	not A
True	False
False	True

`and`

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

*`and` is **strict** — both must be True*

`or`

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

*`or` is **generous** — at least one True*

Exercise 18 Very Easy

Predict the result of each, then verify:

```
>>> not True
>>> not False
```

Your predictions: ____________________ ____________________

Exercise 19 Very Easy

Predict the result of each, then verify:

```
>>> True and True
>>> True and False
>>> False and True
>>> False and False
```

Your predictions: ____________________ ____________________

____________________ ____________________

Exercise 20 Very Easy

Predict the result of each, then verify:

```
>>> True or True
>>> True or False
>>> False or True
>>> False or False
```

Your predictions: ____________________ ____________________

____________________ ____________________

Exercise 21 Medium

Trace through the `not` operator step by step:

```
>>> not (8 > 2)
```

Step 1: Evaluate inside the parentheses: `8 > 2` = __________

Step 2: Apply `not`: `not ___` = __________

Exercise 22 Medium

Trace through `and` step by step:

```
>>> (7 > 4) and (1 > 6)
```

Step 1: Evaluate left side: `7 > 4` = __________

Step 2: Evaluate right side: `1 > 6` = __________

Step 3: Combine: `___ and ___` = __________

Exercise 23 Medium

Trace through `or` step by step:

```
>>> (8 < 3) or (4 == 4)
```

Step 1: Evaluate left side: `8 < 3` = __________

Step 2: Evaluate right side: `4 == 4` = __________

Step 3: Combine: `___ or ___` = __________

Exercise 24 Medium

A library lets you borrow a book if you have a **valid card and** you have **no overdue books**. Predict the output:

```
valid_card = True
overdue_books = 3
can_borrow = valid_card and (overdue_books == 0)
print(can_borrow)
```

Your prediction: ______________________
Now change `overdue_books = 3` to `overdue_books = 0`. What does `can_borrow` become?
____
This is how programs make real decisions using boolean logic.

Exercise 25 **Hard**

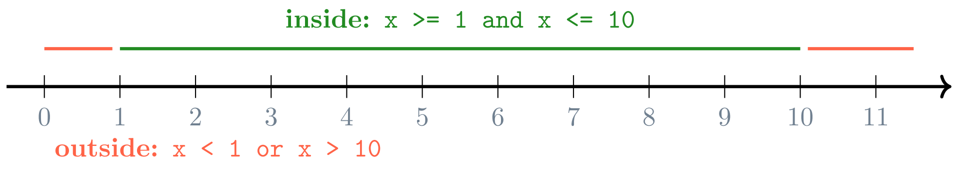
`not` binds first! The `not` operator applies to the **next value only**, before `and`/`or`. Trace through carefully:

```
>>> not True and False
>>> not False or True
```

For the first expression:
Step 1: `not True` = ___________
Step 2: ___ `and False` = ___________
For the second expression:
Step 1: `not False` = ___________
Step 2: ___ `or True` = ___________

5 Compound Boolean Expressions (∼25 min)

Ref: Lecture 4 — Compound Boolean Expressions slides



Exercise 26 **Very Easy**

Predict the result, then verify:

```
>>> x = 5
>>> x >= 1 and x <= 10
```

Your prediction: ______________________
Is `x` inside the range 1 to 10?

Exercise 27 **Very Easy**

Predict the result, then verify:

```
>>> x = 15
>>> x < 1 or x > 10
```

Your prediction: ______________________
Is `x` outside the range 1 to 10?

Exercise 28 Medium

Translate each English phrase into a Python boolean expression. Let `score = 85` and `hours = 10`.

(a) "score is between 60 and 100"

Your expression: ____________________

(b) "hours is above 8 or score is above 90"

Your expression: ____________________

(c) "score is NOT below 50"

Your expression: ____________________

Type each expression in the Shell to verify.

Exercise 29 Medium

Predict the output of this program, then verify:

```python
study_hours = 6
sleep_hours = 9
print(study_hours > sleep_hours)
morning_person = True
has_coffee = False
ready = morning_person and has_coffee
print(ready)
```

Line 3 prints: ____________________ Line 6 prints: ____________________

Exercise 30 Medium

Tricky range mistake! Predict the result for each, with `x = 15`:

```python
>>> x = 15
>>> x < 1 and x > 10
>>> x < 1 or x > 10
```

Your predictions: ____________________ ____________________

Why is the first expression **always** `False` regardless of `x`?

Hint: Can x be less than 1 ***and*** *greater than 10 at the same time?*

Exercise 31 Medium

Fix the bug! A student wrote this code to check if `age` is that of a teenager (between 13 and 19). But it has a logic error:

```python
age = 13
```

```
is_teen = age > 13 and age < 19
print(is_teen)
```

What does it print? ____________________
The student expected `True` (13 **is** a teenager!). What should line 2 be changed to?
Your fix: ____________________
Hint: Think about `>` vs `>=`.

6 Operator Precedence (~15 min)

Ref: Lecture 4 — Operator Precedence slides

Order of Operations

Python evaluates operators in this order (highest priority first):

Priority	Operators	Example
1 (highest)	`**`	`2 ** 3`
2	`* / // %`	`6 / 2`
3	`+ -`	`3 + 4`
4	`< <= > >= == !=`	`x > 5`
5	`not`	`not True`
6	`and`	`a and b`
7 (lowest)	`or`	`a or b`

Remember: arithmetic → comparison → `not` → `and` → `or`

Exercise 32 **Very Easy**

Add parentheses to show the order of evaluation, then predict the result:

```
>>> 5 > 3 and 2 < 4
```

With parentheses: ____________________
Result: __________

Exercise 33 **Very Easy**

Add parentheses to show the order, then predict:

```
>>> not 3 > 5
```

With parentheses: ____________________
Result: __________

Exercise 34 Medium

Trace through this expression step by step:

```
>>> 4 + 1 > 3 and not True
```

Step 1 (arithmetic): `4 + 1` = ___________
Step 2 (comparison): `___ > 3` = ___________
Step 3 (`not`): `not True` = ___________
Step 4 (`and`): `___ and ___` = ___________

Exercise 35 Medium

Careful! `and` binds tighter than `or`. Predict the result:

```
>>> False or True and True
```

Step 1 (`and` first): `True and True` = ___________
Step 2 (`or`): `False or ___` = ___________
What if it were `(False or True) and True` instead?

Exercise 36 Medium

Add parentheses, then evaluate step by step:

```
>>> 7 > 2 and 3 * 2 == 6
>>> not 6 < 1 or 8 > 9
```

First expression result: ___________
Second expression result: ___________
Show your work:

When in Doubt, Use Parentheses!

Parentheses make your code **readable** and remove ambiguity. Even if they're not technically needed, they help:

`not True or False` ← confusing
`(not True) or False` ← crystal clear

7 Truthy, Falsy & String Comparison (~15 min)

Ref: Lecture 4 — Truthy and Falsy / Comparing Strings slides

Falsy (= False)		Truthy (= True)	
`0`	`0.0`	`42`	`-5`
`""`	`None`	`"hello"`	`3.14`
`False`		`True`	

Use `bool()` to check: `bool(0)` → `False`, `bool(42)` → `True`

Exercise 37 — Very Easy

Predict the result, then verify:

```
>>> bool(99)
>>> bool(0)
>>> bool(-3)
>>> bool(0.0)
>>> bool(None)
>>> bool("")
```

Your predictions: ____________________ ____________________

____________________ ____________________

____________________ ____________________

Rule: zero, `0.0`, empty string, `None`, and `False` are Falsy. Everything else is Truthy.

Exercise 38 — Medium

Careful with strings! Predict the result, then verify:

```
>>> bool("python")
>>> bool("")
>>> bool("None")
>>> bool(" ")
```

Your predictions: ____________________ ____________________

____________________ ____________________

Why is `bool("None")` not `False`? Why is `bool(" ")` not `False`?

Exercise 39 — Medium

Truthy in action! Predict the output of this program, then verify:

```python
name = input("Enter your name: ")
if name:
    print(f"Hello, {name}!")
else:
    print("You didn't enter a name.")
```

Run it twice: once entering `Ali`, and once pressing Enter without typing anything.

With `Ali`: ____________________
With empty input: ____________________
An empty string is Falsy, so it acts like ***`False`*** *in an* ***`if`*** *condition.*

Exercise 40 **Very Easy**

Predict the result, then verify:

```
>>> "cat" == "cat"
>>> "cat" == "Cat"
>>> "dog" != "cat"
```

Your predictions: ____________________ ____________________ ____________________
String comparison is case-sensitive!

Exercise 41 **Medium**

Strings are compared character by character (like a dictionary). Predict the result, then verify:

```
>>> "mango" < "orange"
>>> "c" < "d"
>>> "B" < "b"
>>> "Y" < "b"
```

Your predictions: ____________________ ____________________
____________________ ____________________
Uppercase letters always come ***before*** *lowercase:* `"A"` $<$ `"Z"` $<$ `"a"` $<$ `"z"`.

8 Introduction to `if/else` (∼20 min)

Preview — Coming up in Lecture 5

How `if/else` Works

Now that you know how to write boolean expressions, you can use them to make **decisions** in your programs. The `if` statement runs a block of code **only when** a condition is `True`:

```python
if condition:
    # runs when condition is True
else:
    # runs when condition is False
```

Important: The indented lines (4 spaces) are the **body**. Python uses indentation to know which lines belong to the `if` or `else`.

Exercise 42 Very Easy

Type this program into the code editor and run it. Try it **twice**: once entering `25`, and once entering `10`.

```python
age = int(input("Enter your age: "))
if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

When you enter `25`, it prints: ____________________
When you enter `10`, it prints: ____________________
Only ONE branch runs — never both.

Exercise 43 Very Easy

Predict the output, then type it in to verify:

```python
x = 15
if x > 20:
    print("big")
else:
    print("small")
```

Your prediction: ____________________
Now change `x = 15` to `x = 30` and predict again: ____________________

Exercise 44 Very Easy

`if` can be used **without** `else`. Predict the output:

```python
temp = 40
```

```python
if temp > 35:
    print("It is hot today!")
print("Have a nice day")
```

Your prediction:

Now change `temp = 40` to `temp = 20`. What prints now?

The last ***print*** *is* ***not*** *indented, so it runs* ***always****.*

Exercise 45 Very Easy

Predict the output, then verify:

```python
logged_in = False
if not logged_in:
    print("Please log in")
else:
    print("Welcome back")
```

Your prediction: ____________________
Now change `logged_in = False` to `logged_in = True` and predict again: ____________________

The `not` *operator flips the condition.*

Exercise 46 Very Easy

Predict the output, then verify:

```python
color = "red"
if color == "blue":
    print("Sky")
else:
    print("Not sky")
```

Your prediction: ____________________
Now change `color = "red"` to `color = "blue"` and predict again: ____________________

Exercise 47 Very Easy

Predict the output, then verify:

```python
a = 4
b = 9
if a + b > 10:
    print("More than ten")
```

```
else:
    print("Ten or less")
```

Your prediction: ____________________
What is `a + b`? __________ Is it greater than 10? __________

Exercise 48 — Very Easy

Predict the output:

```
raining = True
if raining:
    print("Take an umbrella")
else:
    print("Enjoy the sun")
```

Your prediction: ____________________
Now change `raining = True` to `raining = False` and predict again: ____________________
A boolean variable can be used directly as the condition.

Exercise 49 — Very Easy

Predict the output, then verify:

```
marks = 85
if marks >= 50:
    print("You passed!")
    print("Well done!")
print("Goodbye")
```

Your prediction:

Now change `marks = 85` to `marks = 30`. Which lines still print?

*Both indented lines belong to the `if` body. `"Goodbye"` is not indented, so it runs **always**.*

9 Capstone Challenges (∼20 min)

These problems combine everything you have learned. Write your solutions in Thonny's code editor.

Exercise 50 Medium

Movie Ticket Discount. Write a program that:

1. Asks the user for their age
2. Prints whether they get a child discount (age 12 or below)
3. Prints whether they get a senior discount (age 60 or above)

Example (user enters `8`):

```
Child discount: True
Senior discount: False
```

Example (user enters `65`):

```
Child discount: False
Senior discount: True
```

Hint: Use two separate comparison expressions.
Write your code:

Exercise 51 Medium

Password Length Check. Write a program that:

1. Asks the user for a password
2. Prints whether the password is at least 8 characters long
3. Prints whether the password is between 8 and 20 characters (inclusive)

Example (user enters `hello123`):

```
Long enough: True
Valid length: True
```

Write your code:

Exercise 52 Medium

Grade Check. Write a program that:

1. Asks the user for their marks (out of 100)
2. Prints whether their marks are passing (50 or above)
3. Prints whether their marks qualify for distinction (75 or above)

Example (user enters `82`):

```
Passing: True
Distinction: True
```

Write your code:

Exercise 53 Hard

Leap Year. A year is a leap year if:

- It is divisible by 4, **AND**
- It is NOT divisible by 100, **OR** it is divisible by 400

Write a program that:

1. Asks the user for a year
2. Computes a boolean expression for whether it's a leap year
3. Prints the result

Example (user enters `2024`):

```
2024 is a leap year: True
```

Example (user enters `1900`):

```
1900 is a leap year: False
```

Hint: Use `%` (modulo). A number is divisible by n if `x % n == 0`.
Write your code:

Exercise 54 **Hard**

Triangle Validator. Three side lengths form a valid triangle if and only if the sum of any two sides is greater than the third side.
Write a program that:

1. Asks the user for three side lengths
2. Prints whether they form a valid triangle

Example (user enters `3`, `4`, `5`):

```
Valid triangle: True
```

Example (user enters `1`, `2`, `10`):

```
Valid triangle: False
```

Hint: You need three conditions combined with `and`.
Write your code:

Self-Assessment Checklist

Before you leave, check off what you can do:

- ☐ I can use `print()` with commas and concatenation, and I know the difference
- ☐ I know that `input()` always returns a `str` and I must cast it for arithmetic
- ☐ I can use f-strings to format output with expressions
- ☐ I know that `bool` has exactly two values: `True` and `False` (case-sensitive)
- ☐ I can use all six comparison operators: `==`, `!=`, `<`, `>`, `<=`, `>=`
- ☐ I know the difference between `=` (assignment) and `==` (comparison)
- ☐ I can use `not`, `and`, and `or` and I know their truth tables
- ☐ I can write compound boolean expressions for range checks
- ☐ I know the full operator precedence: arithmetic → comparison → `not` → `and` → `or`
- ☐ I know which values are Truthy and which are Falsy

☐ I can compare strings (case-sensitive, lexicographic order)

☐ I can write a basic `if/else` statement and I understand that indentation defines the body

☐ When in doubt, I use **parentheses** to make my expressions clear