Class

4

CSC 141 Computer Science I

Making Decisions: Boolean Expressions & if / else

Dr. Si Chen (schen@wcupa.edu)

Tue / Thu · 11:00 AM – 12:15 PM · Mitchell Hall 102
Fall 2026 · Week 3 · Tuesday, September 8, 2026
PA1 now due Thu Sep 17 · Quiz 1 moved to Tue Sep 15 ·
wcu.ninja/csc141

Recap from last class

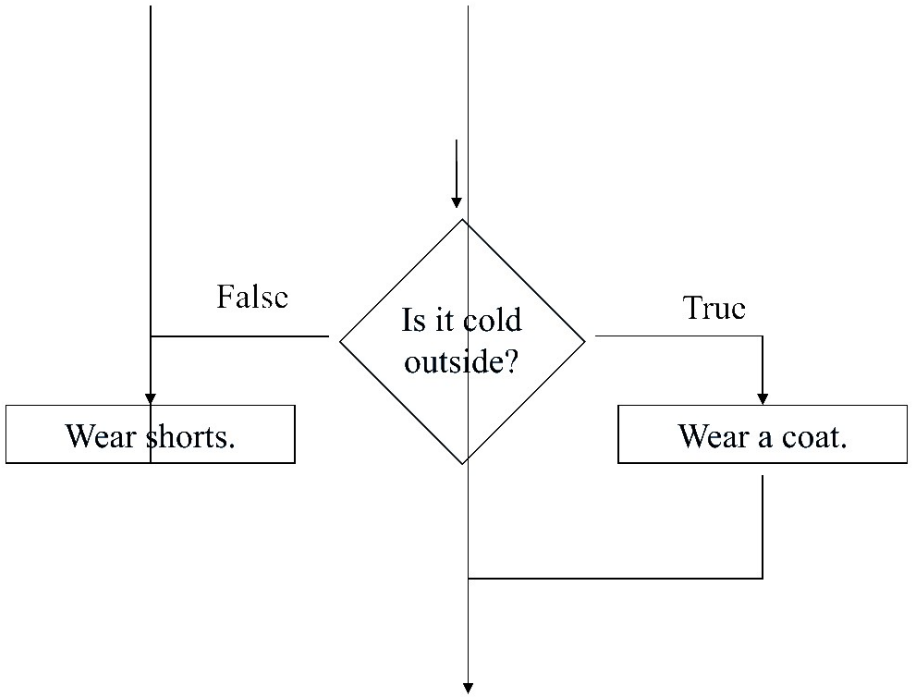
- A **variable** is a named box; `=` assigns, `+=` updates, `a, b = b, a` swaps.
- Types: `int`, `float`, `str`, `bool`; convert on purpose with `int()` / `float()` / `str()`.
- **f-strings** build clean output: `f"{celsius:.1f}"`, `f"{name:>10}"`.
- Every program so far ran **top to bottom, every line, every time**. Today: programs that **choose**.
- **Two schedule changes: PA1 is now due Thu Sep 17, 11:59 PM**, and **Quiz 1 moves to Tue Sep 15**.

PART 1

Programs that make decisions

The if statement: run some code only when a condition is true

Is it cold outside?



- A **decision** asks a yes/no question and takes **one of two paths**.
- **Flowchart**: the diamond is the question; the boxes are actions; both paths rejoin below.
- In code the question is a **Boolean expression** — it evaluates to `True` or `False`.
- Same example, in Python, in a moment: `if cold_outside: ... else: ...`

From my Class 9 deck (2018) — the flowchart is language-independent; only the syntax changes.

bool: the type with two values

```
is_raining = True
is_weekend = False
print(type(is_raining))   # <class 'bool'>

# comparisons PRODUCE bool values:
age = 19
print(age > 18)             # True
print(age == 18)          # False  (== compares)
print(age != 18)            # True

is_adult = age >= 18        # store the answer
print(is_adult)             # True
```

- `bool` is the fourth basic type: exactly two values, `True` and `False` (capital letters!).
- **Comparison operators** produce a bool: `<` `>` `<=` `>=` `==` `!=`.
- `==` **compares**; `=` **assigns**. Mixing them up is the #1 decision bug.
- You can **store** a comparison result in a variable — that is a **flag** (more Thursday).

The six comparison operators

Operator → meaning

- `x < y` — less than
- `x > y` — greater than
- `x <= y` — less than or equal
- `x >= y` — greater than or equal
- `x == y` — **equal** (two `=` signs)
- `x != y` — not equal

- Each one **evaluates to True or False** — nothing else.

Try it in the playground

```
>>> x = 5
>>> y = 10
>>> x > y
False
>>> x == 5
True
>>> x < y
True
>>> 3.0 == 3
True          # same value
>>> "5" == 5
False         # str vs int!
```

PART 2

if and else

Syntax, the colon, and why indentation is not optional in Python

The if statement

```
temp = float(input("Temperature in F? "))

if temp > 100:
    print("It is really hot!")  # ONLY if True
    print("Stay hydrated.")     # same block

print("Have a nice day.")  # not indented: always
```

- `if` + a **condition** + a **colon** `:`.
- The **indented block** underneath runs only when the condition is `True`.
- The block ends where the indentation ends — **no braces, no end**.
- Convention: **4 spaces** per level. VS Code does it for you after the colon.

Try it: 105 → three lines print; 70 → only "Have a nice day."

Indentation IS the block — the "Hodor" bug

```
cold_outside = False

# What the programmer MEANT:
if cold_outside:
    print("Wear a coat.")
    print("Hodor!")

# What they TYPED (one space too few):
if cold_outside:
    print("Wear a coat.")
print("Hodor!")   # outside the if: ALWAYS prints
```

- In Java this bug came from **missing braces**: only the next statement belonged to the `if`.
- In Python the block is **exactly the indented lines** — one un-indented line silently leaves the block.
- Python **refuses** inconsistent indentation (`IndentationError`) — read the error, fix the spacing.
- **Never mix tabs and spaces.** Let VS Code insert 4 spaces.

if / else: two paths, exactly one runs

```
temp = float(input("Temperature in F? "))

if temp > 100:
    print("It is really hot!")
else:
    print("It is not that hot.")

print("Have a nice day.")  # rejoin: always runs
```

- `else:` has **no condition** — it is "otherwise".
- `if` and `else` line up at the **same indentation**; each has its own indented block.
- **Exactly one** of the two blocks runs. Never both, never neither.
- After the decision the two paths **rejoin** — the flowchart's bottom arrow.

Comparing strings

Rules

- `==` compares **text exactly**: `"Ada" == "ada"` is `False` — **case matters**.
- `<` and `>` compare **alphabetically**… by character **code**: `"Ada" < "Bob"` is `True`.
- Surprise: `"Zoe" < "ada"` is `True` — **uppercase letters come before lowercase** (A = 65, a = 97).
- Case-insensitive check: convert first — `answer.lower() == "yes"`.
- Java needed `.equals()` / `equalsIgnoreCase()`; Python just uses `==` on the text.

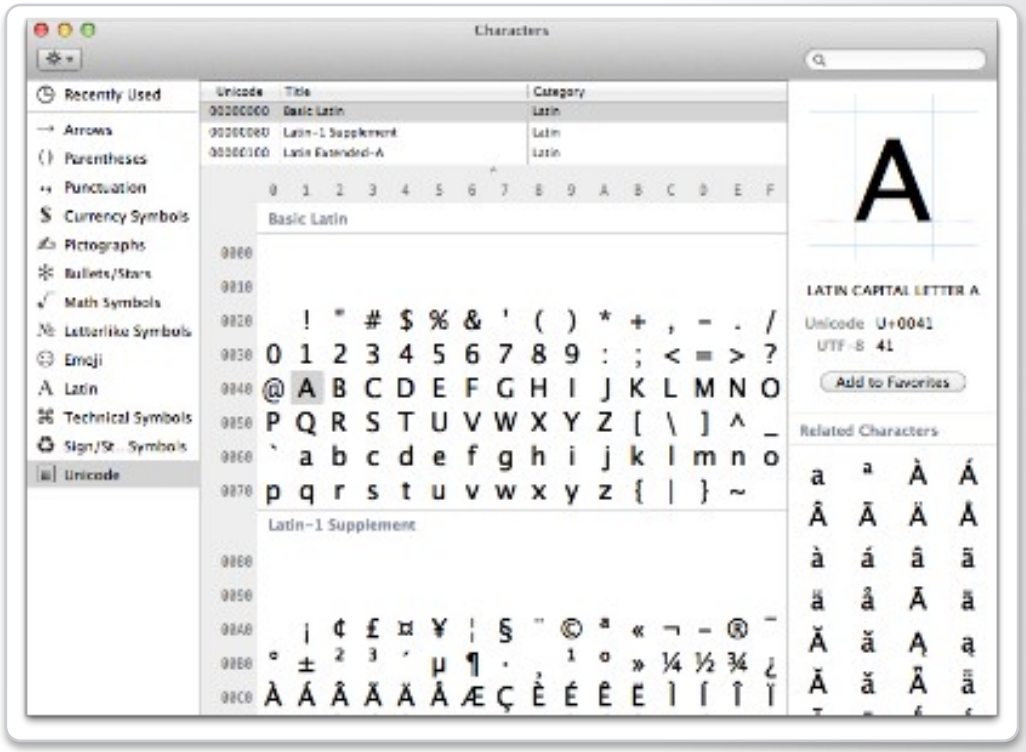
string_compare.py

```
password = input("Password? ")
if password == "open sesame":
    print("Access granted.")
else:
    print("Access denied.")

secret = input("Secret word? ")
if secret.lower() == "wcupa":
    print("Welcome, Golden Ram!")

print("Ada" < "Bob")   # True
print("Zoe" < "ada")   # True (!)
```

Why "Zoe" < "ada": characters are numbers



- Every character has a **code** (Unicode): `A` is 65, `B` is 66 … `Z` is 90; `a` is 97 … `z` is 122.
- Python compares strings **one character at a time** using these codes.
- `ord("A")` → 65 and `chr(65)` → `"A"` let you see the codes yourself.
- That is why sorting "raw" text puts all capitals first — normalize with `.lower()` when order should be human-friendly.

macOS Character Viewer (my 2018 slide): "A" is U+0041 = 65.

PART 3

and, or, not

Combining conditions — and the shortcut that makes Python read like English

Three logical operators

and

`A and B` is `True` only if **both** are `True`.

```
temp > 70 and not raining
```

T and T → T

T and F → F

F and T → F

F and F → F

or

`A or B` is `True` if **at least one** is `True`.

```
temp > 95 or temp < 20
```

T or T → T

T or F → T

F or T → T

F or F → F

not

`not A` **flips** the value.

`not (temperature > 100)` → "below the maximum"

not T → F

not F → T

Reads like English: `if not raining:`

The loan check: and vs or

```
MIN_SALARY = 30000
MIN_YEARS = 2
salary = float(input("Salary? "))
years = int(input("Years on the job? "))

# BOTH requirements must hold:
if salary >= MIN_SALARY and years >= MIN_YEARS:
    print("You qualify for the loan.")
else:
    print("You do not qualify.")

# a generous bank: EITHER is enough
if salary >= MIN_SALARY or years >= MIN_YEARS:
    print("Pre-approved!")
```

- The 2018 loan example: salary ≥ 30,000 **and** ≥ 2 years on the job.
- Swap `and` for `or` and the policy changes completely — read conditions **aloud** to check them.
- **Short-circuit**: `and` stops at the first `False`; `or` stops at the first `True`. The right side may never run.
- Named constants (`MIN_SALARY`) make the rule readable and easy to change — PA1 habit.

Range checks and precedence

Rules of thumb

- "Is `x` between 1 and 100?" → `x >= 1 and x <= 100`
- Python shortcut, reads like math: `1 <= x <= 100` (a **chained comparison**).
- **Wrong** (from Java habits): `1 <= x <= 100` is fine in Python but `x == 1 or 2` is **not** what you think — `2` alone is `True`, so it is always `True`!
- **Precedence**: `not` binds tighter than `and`, which binds tighter than `or`. Comparisons happen before all three.
- When in doubt: **parentheses**. They cost nothing and prevent bugs.

Watch the traps

```
x = 42
print(x >= 1 and x <= 100)   # True
print(1 <= x <= 100)         # True

# the classic trap:
grade = "C"
print(grade == "A" or "B")   # "B"  -> truthy!
print(grade == "A" or grade == "B")  # False

# precedence: and before or
print(True or False and False)   # True
print((True or False) and False) # False
```

"Almost right": where AI code fails

84%

of developers use or plan to use AI tools in their workflow

Stack Overflow Developer Survey 2025

46%

actively distrust the accuracy of AI tool output (only 33% trust it)

Stack Overflow Developer Survey 2025

66%

say their top frustration is AI answers that are "almost right, but not quite"

Stack Overflow Developer Survey 2025

An `if` with `>` instead of `>=`, an `or` where `and` was meant, a comparison against `"B"` instead of `grade == "B"` — these all run without errors and look right. **Boundary conditions are where "almost right" code fails**, and the only defense is a human who can read the condition and test the edge (exactly 100? exactly 18?).

Hands-on: two small decisions

HANDS-ON ⏱ 10 min

- Open **wcu.ninja/csc141/python?class=4**. Plan first, then code.
- **A. Movie ticket** (`age_gate.py`): ask for age; under 13 → $8.00, otherwise $12.00; print age and price with an f-string (`:.2f`).
- **B. Login check** (`password_check.py`): ask username and password; grant access only if **both** match stored constants; otherwise say "Invalid username or password." (never say which one was wrong).
- **Stretch:** make the username check case-insensitive but keep the password case-sensitive.
- **Test the boundary:** age 12 and age 13 — which price?

Skeleton:

```
age = int(input("How old? "))
if age < 13:
    price = CHILD_PRICE
else:
    price = ADULT_PRICE
print(f"... ${price:.2f}")
```

B: `if user == STORED_USER and pw == STORED_PASSWORD:`

Stretch: `user.lower() == STORED_USER`

Debrief — what you practiced

- **Comparisons** produce `bool`; `==` compares, `=` assigns.
- `if` / `else` with a colon and an **indented block** — indentation is the syntax.
- Strings compare by **character code**; normalize with `.lower()` when case should not matter.
- `and` / `or` / `not` combine conditions; chained comparisons `1 <= x <= 100` read like math.
- You tested a **boundary** (12 vs 13) on purpose — the habit that catches "almost right" code.

Schedule change — and before Thursday

- **PA1 is now due Thursday, September 17, 11:59 PM** — one extra week. Nothing else changed: same six files, same output formats. The updated handout is on the course site.
- **Quiz 1 moves to Tuesday, September 15** (first 15 minutes). Because it is now after Thursday's class, it covers **Classes 1–5**: types, `input()` conversion, expressions, f-strings, comparisons, `if`/`elif`/`else`, `and`/`or`/`not`. Closed notes, no AI.
- Practice: rewrite today's ticket program so seniors (65+) also pay $8.00. Which operator did you need?
- Thursday: **elif chains, nested decisions, and designing a program before you code it**.
- Course site: **wcu.ninja/csc141** — Class 4 code and playground link are posted.

Sources

- Flowcharts, "cold outside" example, Check.java, loan-qualification and secret-word examples: Dr. Si Chen, CSC 141 Spring 2018 Class 9–10 decks (Java), translated to Python.
- Python Language Reference §6.10 (comparisons, chained comparisons) and §6.11 (Boolean operations, short-circuit).
- Python Tutorial §4.1 (if statements) and PEP 8 (4-space indentation; never mix tabs and spaces).
- Stack Overflow, "2025 Developer Survey — AI" (84% use/plan to use; 46% distrust accuracy vs 33% trust; 66% frustrated by almost-right answers).
- Unicode Standard, Basic Latin block (U+0041 "A" = 65; U+0061 "a" = 97).
- Gaddis, Starting Out with Java, Ch. 3 (Decision Structures) — original source of several 2018 examples.