Decision Logic - From C# to Python

CSCI-2910 — Server-Side Web Programming | Practice Handout

You already know how to make decisions in code — you've written `if`, `else if`, `else`, and `switch` in C# for two semesters. This handout is about translating that instinct into Python's syntax, then using it to finish a program that isn't done yet.

Part 1 — Quick Reference

Keep this open while you work.

Concept	C#	Python
Basic decision	`if (x > 5) { ... }`	`if x > 5:`
Block delimiter	`{ }`	Indentation (4 spaces) after `:`
Else if	`else if (x > 0) { ... }`	`elif x > 0:`
Else	`else { ... }`	`else:`
AND	`&&`	`and`
OR	`||`	`or`
NOT	`!`	`not`
Equality check	`==`	`==`
Multi-branch	`switch (x) { case ...: ... break; }`	`match x: case ...:`
Default case	`default:`	`case _:`

A few things that trip up C# developers the first week:

- Python has **no braces**. The colon `:` opens a block; the next line's indentation is what closes it.
- There is **no `break`** in a Python `match` — each `case` stops on its own after running.
- Comparing with `=` instead of `==` is a mistake in both languages, but Python won't even let you use `=` inside an `if` condition — it's a syntax error, not a silent bug, which is actually a small safety net C# doesn't give you either.

Part 2 — Translate These

For each C# snippet, write the Python equivalent in the blank underneath. Don't run these yet — just translate by hand first.

1.

```
if (temperature >= 90)
{
    Console.WriteLine("It's hot.");
}
else
{
    Console.WriteLine("It's manageable.");
}
```

Python:

```

```

2.

```
if (gpa >= 3.5)
{
    Console.WriteLine("Dean's List");
}
else if (gpa >= 3.0)
{
    Console.WriteLine("Honor Roll");
}
else
{
    Console.WriteLine("Keep working!");
}
```

Python:

```

```

3.

```
if (hasTicket && age >= 18)
{
    Console.WriteLine("Admitted.");
}
```

Python:

```

```

4.

```
if (!isRaining || hasUmbrella)
{
    Console.WriteLine("Let's walk.");
}
```

Python:

```

```

Part 3 — Finish the Program: *Movie Night Picker*

Below is a partially working Python program. It collects a few facts from the user and is *supposed* to recommend how to spend movie night — but several pieces are missing. Your job is to fill in the `# TODO` blocks. The comments tell you exactly what logic belongs there; the syntax is up to you.

Read the whole program before you start writing — later TODOs depend on variables set up earlier.

```
genre = ''
minutes_available = -1
snacks_in_house = ''
mood = ''

print("---------------------- Movie Night Picker ----------------------")
print("  Answer a few questions and I'll help you plan movie night.")
print("----------------------------------------------------------------")

while genre.strip() == '':
    genre = input("  1. Favorite genre tonight (comedy, horror, drama,
action): ")

while minutes_available == -1:
    try:
        minutes_available = int(input("  2. Minutes available tonight: "))
        if minutes_available < 0:
            minutes_available = -1
            print("  That's not a real number of minutes. Try again.")
    except ValueError:
        print("  Numbers only, please!")

while snacks_in_house.strip() == '':
    snacks_in_house = input("  3. Do you have snacks in the house? (yes/no):
")

while mood.strip() == '':
    mood = input("  4. Are you in the mood to focus, or just relax?
(focus/relax): ")

print("\n----------------------- Facts Collected -----------------------")
print(f"  Genre:              {genre}")
print(f"  Minutes available:  {minutes_available}")
print(f"  Snacks in house:    {snacks_in_house}")
print(f"  Mood:                {mood}")
print("----------------------------------------------------------------")

# TODO #1 — Length category
# Using if / elif / else, set a variable called `length_category` based on
# minutes_available:
#   < 30            -> "short film"
#   30 to 89         -> "standard movie"
#   90 or more        -> "epic / miniseries night"
```

```
# (Decide for yourself whether the boundaries are inclusive or exclusive,
# but be consistent, and be ready to explain your choice.)




# TODO #2 — Snack run decision
# Using and / or / not, decide whether a snack run is needed.
# A snack run IS needed if the user does NOT have snacks in the house
# AND they are watching something 30 minutes or longer.
# Store the result (True or False) in a variable called `snack_run_needed`.




# TODO #3 — Genre suggestion, with match
# Replace the logic below with a match statement on `genre.lower()` that
# prints ONE suggested title per genre. Include at least four genres
# (comedy, horror, drama, action) plus a default case using `_` for
# anything else the user might type.




# TODO #4 — Nested conditional
# If mood is "focus", recommend movies from `length_category` "standard
movie"
# or "epic / miniseries night" only — print a message explaining why a
# "short film" isn't a great fit for focused viewing.
# If mood is "relax", any length_category is fine — just print the
# length_category as the recommendation.
# This should be a nested if: check mood first, then (when appropriate)
# check length_category inside it.




print("\n------------------------------ Plan -------------------------------")
# TODO #5 — Final summary
# Print a short summary that includes: the length_category, whether a
# snack run is needed, and the mood-based recommendation from TODO #4.
# Use an f-string.


print("-------------------------------------------------------------------")
```

Checklist

- ◯ TODO #1 uses `if` / `elif` / `else` — no bare `if` chains without `elif`
- ◯ TODO #2 uses at least one `and`, `or`, or `not`
- ◯ TODO #3 is a genuinely **nested** conditional, not two separate `if` blocks
- ◯ The program runs top to bottom without a `NameError` or `IndentationError`
- ◯ You added at least one comment of your own explaining a section in your words

---

Part 4 — Stretch Challenge (optional)

Everything above lives in one long block of top-level code. Try pulling **one** piece of logic — for example, the genre suggestion from TODO #3 — out into its own function:

```
def suggest_movie(genre):
    # move your match statement in here
    # return the suggestion instead of printing it directly
    ...
```

Then call it from the main part of your program: `suggestion = suggest_movie(genre)`.

This is a preview of next class: breaking a program into reusable pieces instead of one long script — the same instinct as pulling logic into a method in C#, just without the class wrapper required to do it.