if-else, if-elif-else & Logical Operators

AiCE Software Engineering Foundations — Python Programming

A plain if statement can only do something or do nothing. This lesson adds the missing pieces — else for a guaranteed alternative path, elif for choosing between several outcomes, and the logical operators that let a single condition check more than one thing at once.

1. The if-else Statement

if-else adds a second path — the else block — that runs when the condition is False, guaranteeing that something always happens either way:

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

```
score = float(input("Enter your entrance exam score: "))

if score >= 60:
    print("ADMISSION STATUS: ACCEPTED")
    print(f"Score: {score}/100 – Well done!")
else:
    print("ADMISSION STATUS: NOT ACCEPTED")
    print(f"Score: {score}/100 – Minimum required: 60")
```

Running this with a score above 60 and again with a score below 60 sends the program down a completely different path each time, based on that single condition.

> **Conditionals Give Code the Ability to Think**
> *A program that always does the same thing regardless of the situation is not intelligent — it's a template. Conditionals are what give code the ability to respond differently depending on what it's given.*

2. The if-elif-else Statement

When there are more than two possible outcomes, elif — short for 'else if' — adds additional conditions checked in order after the initial if. Python evaluates conditions top to bottom and executes only the first block whose condition is True:

```
if condition_one:
    # runs if condition_one is True
elif condition_two:
    # runs if condition_one is False AND condition_two is True
else:
    # runs if ALL conditions above are False
```

A grade classification system with several possible outcomes:

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

if score >= 70:
    grade = "A"
elif score >= 60:
    grade = "B"
elif score >= 50:
    grade = "C"
elif score >= 40:
    grade = "D"
else:
    grade = "F"

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

Top-Down, First-Match Evaluation

When a score of 85 is entered, Python hits score >= 70 first and executes that block immediately — it never even checks the elif or else conditions below. This top-down, first-match evaluation is fundamental to how if-elif-else works. The else block at the end is optional but recommended, acting as a safety net for every case the conditions above don't cover.

3. Logical Operators — and, or, not

Sometimes a single comparison isn't enough. Logical operators combine multiple conditions into a single expression:

Operator	Meaning	Returns True When
and	Both conditions must be True	condition_a AND condition_b are both True
or	At least one must be True	condition_a OR condition_b (or both) are True
not	Reverses the boolean result	the condition is False (not True = False)

Using and — Both Conditions Must Be True

```
# Scholarship requires: score >= 75 AND age <= 30
score = float(input("Score: "))
age   = int(input("Age    : "))

if score >= 75 and age <= 30:
    print("Eligible for the AiCE Scholarship.")
else:
    print("Does not meet the scholarship criteria.")
```

Using or — At Least One Condition Must Be True

```
is_student = input("Are you a student? (yes/no):
").strip().lower()
is_staff   = input("Are you a staff member? (yes/no):
").strip().lower()

if is_student == "yes" or is_staff == "yes":
    print("Access granted.")
else:
    print("Access denied.")
```

Access is granted as long as at least one answer is 'yes' — only when both are 'no' is access denied.

Using not — Reversing a Condition

```
is_verified = False

if not is_verified:
    print("Account not verified. Please check your email.")
else:
    print("Account verified. Welcome back.")
```

not is_verified evaluates as not False, which is True, so the first block runs — not is a clean way to check for the absence of something rather than its presence.

Key Takeaways

else guarantees a fallback path, and elif lets a program choose between several outcomes, evaluated strictly top to bottom with only the first matching block running. The logical operators — and requiring both conditions, or requiring at least one, and not reversing a single condition — let you build genuinely nuanced decision logic from simple comparisons. The next lesson covers nesting conditionals inside one another, comparing strings directly in conditions, and a compact one-line shorthand for simple if-else logic.