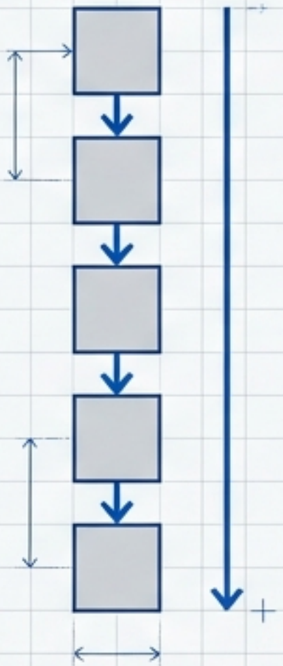
Constructing Intelligent Programs: Your Guide to Python's Conditional Logic

The Architect's Toolkit for Control Flow

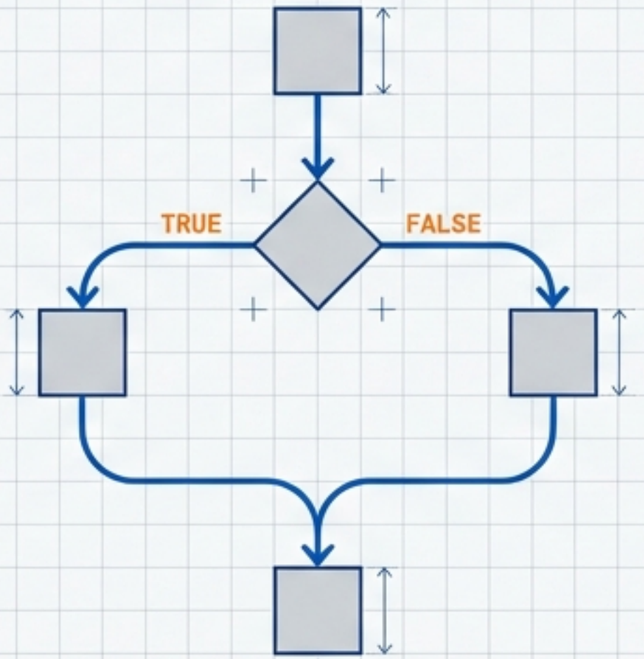
Code by Default is a Straight Line. Intelligence Requires Pathways.

By default, a Python script executes commands one after another, from top to bottom. This is predictable but limited. To create programs that can react, decide, and adapt, we must move beyond this linear path. We need to build junctions, gateways, and decision points into our code. This is called **Control Flow**. It is the art of directing the execution path based on specific conditions.

Linear Execution



Conditional Execution



Tool 1: The Foundational Gateway — The `if` Statement

The `if` statement is the most basic tool for control flow. It tests a condition and executes a block of code ***only if*** that condition is True. If the condition is False, the block is simply skipped.

```
if condition:
    # Code to execute if the condition is True
    statement(s)
```

Annotated Example

```
# Code Snippet
age = 18

if age >= 18:
    print("Eligible")
```

The Test: This expression evaluates to either True or False.

The Consequence: This indented block only runs if The Test is True.

Tool 2: The Two-Way Junction — The `if-else` Statement

Purpose

When a single path isn't enough, `if-else` provides an alternative.

The `else` block runs *only when* the `if` condition is False, is False, guaranteeing that one of the two blocks will always execute.

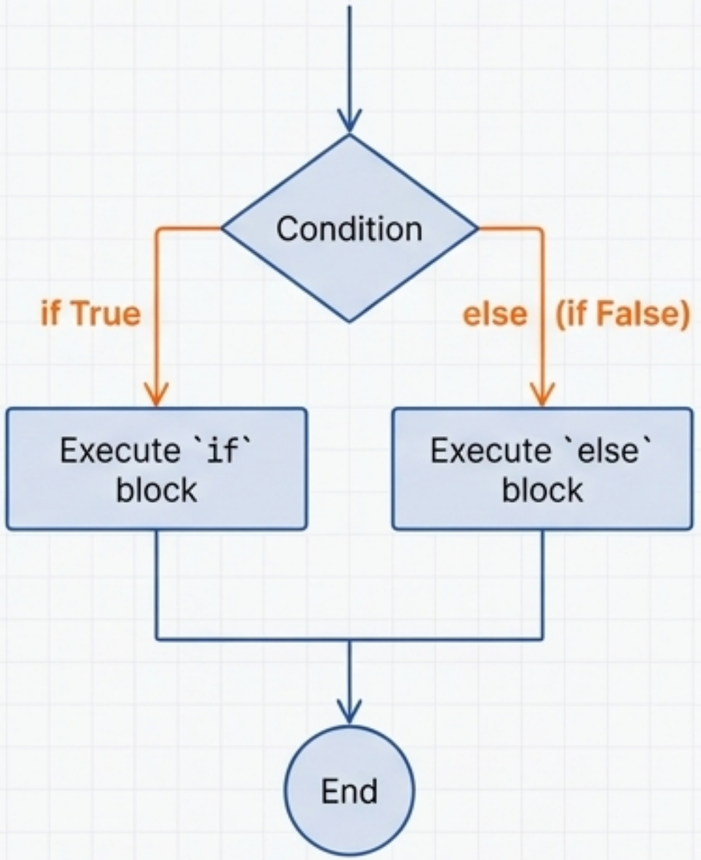
Syntax Schematic

```
if condition:
    # Executes if condition is True
    statement(s)
else:
    # Executes if condition is False
    statement(s)
```

Annotated Example

```
# Code Snippet
marks = 40

if marks >= 33:   # <-- This is True
    print("Pass") # <-- This block executes
else:             ←┐
    print("Fail") # <-- This block is skipped
```

Flowchart



Tool 3: The Multi-Path Interchange — The `if-elif-else' Statement

Purpose

Used to check several conditions in a specific order. Python executes the block for the *first* condition that evaluates to `True` and skips all others. The final `else` acts as a default case if no conditions are met.

Syntax Schematic

```
if condition_1:
    statement(s)
elif condition_2:
    statement(s)
else:
    statement(s)
```

Annotated Example

```
# Code Snippet
score = 85

if score >= 90:
    print("A")
elif score >= 75:
    print("B")
else:
    print("C")
```

Check 1: `85 >= 90` is False. Move on.

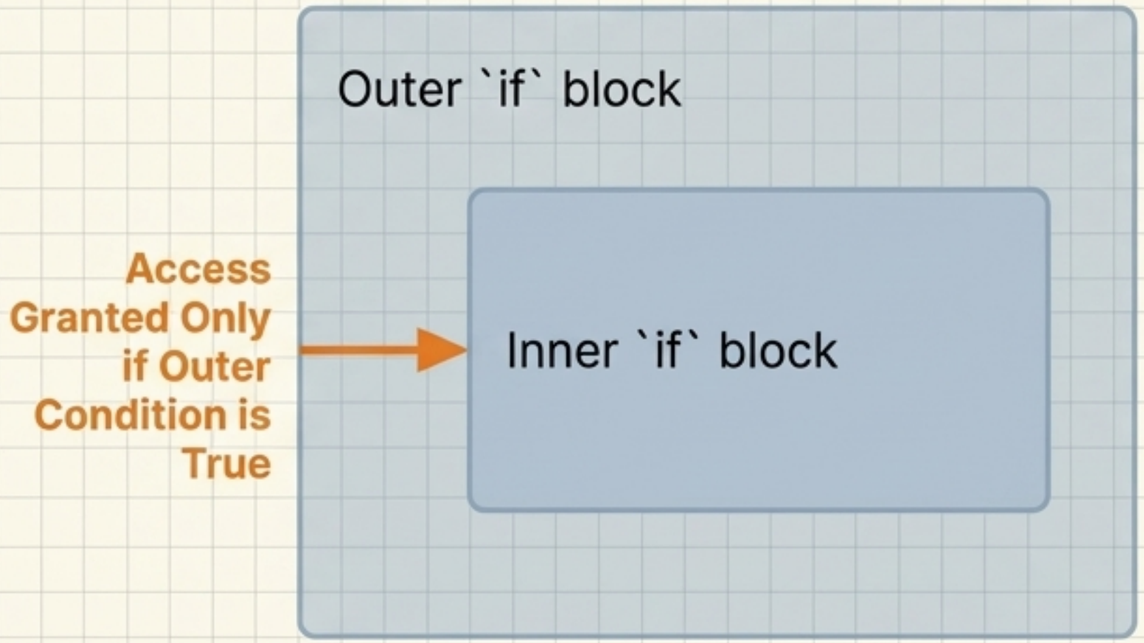
Check 2: `85 >= 75` is True. Execute this block.

Skipped: A previous condition was met.

Advanced Blueprinting: Nesting Logic for Complex Criteria

Purpose

Sophisticated logic often requires checking a condition *after* a previous condition has already been met. By placing a conditional statement inside another, we create ‘nested’ logic. The inner block is only reachable if the outer block's condition is True.



Annotated Example

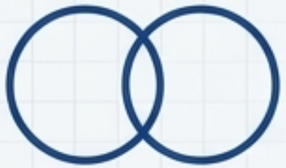
```
# Code Snippet
age = 20
is_citizen = True

if age >= 18:
    if is_citizen:
        print("Eligible to vote")
```

This entire inner block is only considered because `age >= 18` was True.

The **inner** check is performed.

Precision Tools: Creating Compound Rules with Logical Operators

Sometimes, a single check is insufficient. Logical operators allow you to combine multiple conditions into a single, more expressive statement.



and Operator: Requires all conditions to be True.

```
if age > 18 and age < 60:
```

This checks if the age is simultaneously greater than 18 *and* less than 60.

or Operator: Requires at least one condition to be True.

```
if city == "Lucknow" or city == "Delhi":
```

This checks if the city is Lucknow *or* if it is Delhi.

The Craftsman's Touch: Elegant and Readable Range Checks

A hallmark of Python is its emphasis on clean, readable code. For checking if a value falls within a range, Python allows you to "chain" comparisons together in a way that mirrors mathematical notation.

The Standard Method (using `and`)

```
if x > 1 and x < 10:
    print("Valid number")
```

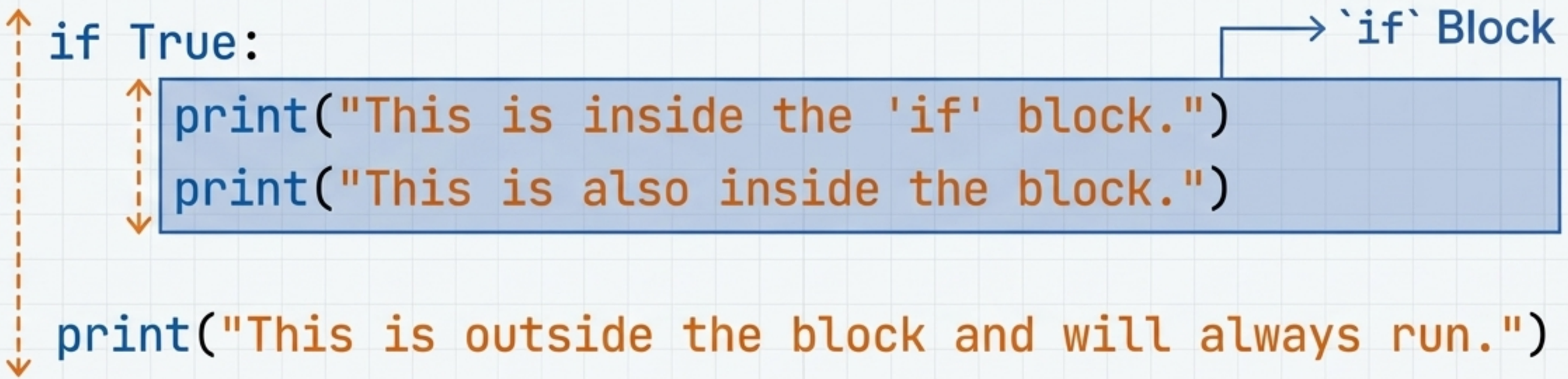
The Pythonic Method (Chained Comparison)

```
if 1 < x < 10:
    print("Valid number")
```

Both achieve the same result, but the chained version is more concise and widely considered more readable.

The Rule of Construction: Indentation Defines Structure

Unlike many languages that use brackets (`{}`), Python uses whitespace to define code blocks. The level of indentation is not optional or for style; it is part of the syntax and dictates which lines of code belong to which statement.



All lines indented at the same level after a statement (like `if` or `else`) form a single execution block.

Structural Placeholders: Using `pass` for Future Development

Purpose

What if you need to define a conditional block but are not ready to write the logic for it yet? An empty block would cause a syntax error. The `pass` statement is a null operation. It does nothing. It simply acts as a placeholder where a statement is syntactically required, allowing your program to run without error.

Use Case

Ideal for stubbing out functions or conditional branches that you intend to implement later.

```
# Code Snippet
x = 5

if x > 0:
    pass # TODO: Implement logic
         for positive numbers later
else:
    print("Number is not
    positive.")
```

Case Study: Assembling the Tools for a Thermostat

The Challenge

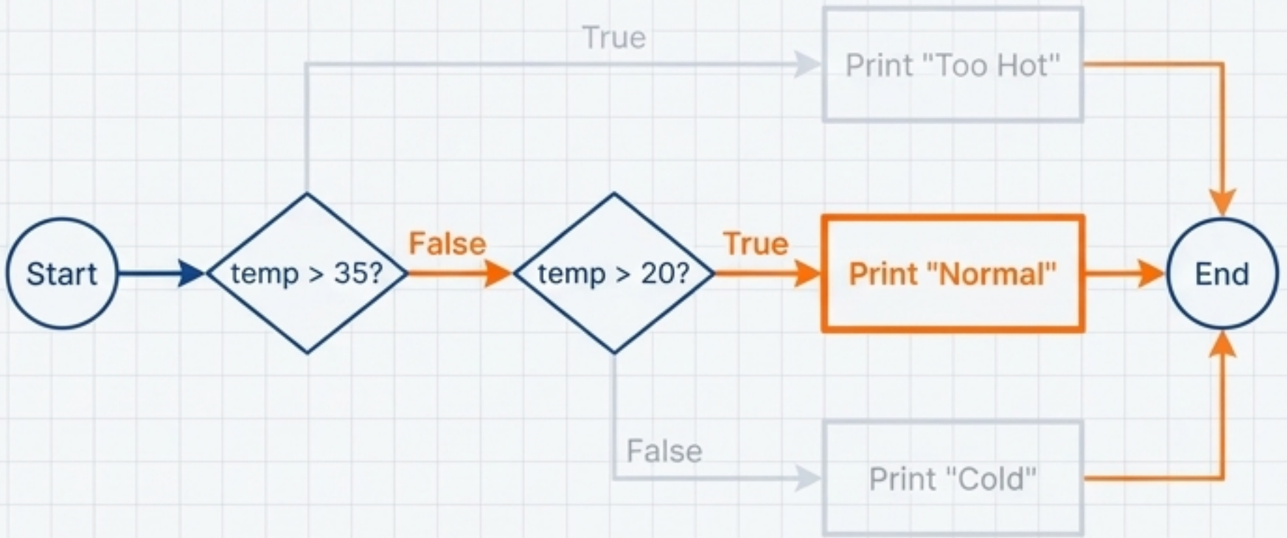
Write a simple program that provides feedback based on a given temperature.

The Blueprint (Code)

```
temperature = 30

if temperature > 35:
    print("Too Hot")
elif temperature > 20:
    print("Normal")
else:
    print("Cold")
```

Execution Analysis

1. `temperature > 35` is checked. `30 > 35` is **False**
2. The program moves to the `elif` branch.
3. `temperature > 20` is checked. `30 > 20` is **True**
4. The code block `print("Normal")` is executed.
5. All subsequent branches (`else`) are skipped.



Your Python Architect's Toolkit: A Summary

Tool: `if`

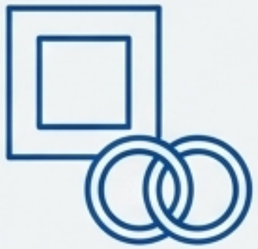
The simple gateway. Executes code for a single, `True` condition.

Tool: `if-else`

The two-way junction. Chooses between two exclusive paths.

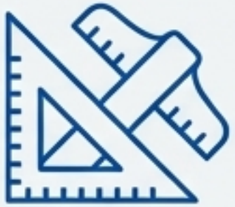
Tool: `if-elif-else`

The multi-path interchange. Selects the first true path from an ordered series.



Technique: **Nesting & Operators**

For crafting complex, layered, and precise logical rules.



Principle: **Indentation &** `pass`

The non-negotiable rules for defining structure and placeholders.

Beyond the Blueprint

Control flow is more than just syntax; it is the art of teaching a program how to think. The conditional tools you've learned are the foundation upon which all complex algorithms, applications, and intelligent systems are built.

Master these tools, and you can construct anything.