Python Conditional Statements (`if`, `elif`, `else`)

Python uses conditional statements (`if`, `elif`, `else`) to execute code blocks only when specified conditions are true. These statements control program flow: the `if` keyword checks a Boolean expression, and if it evaluates to `True`, its indented block (the "suite") runs. You can follow an `if` with any number of `elif` ("else if") clauses and an optional `else` clause. Python evaluates each `if`/`elif` test in order and executes *only the first* matching block; if none match, the `else` block (if present) runs. For example, the official Python tutorial shows an `if/elif/else` sequence handling different integer cases, with only the first true branch executing.

`if` Statement Syntax

```
if <condition>:
    <indented code block>
```

The `<condition>` is any expression that yields `True` or `False` (for example, comparisons like `x > 0`, membership tests like `'a' in my_list`, or combined tests with `and`/`or`). If the condition is `True`, Python executes the indented block; if it is `False`, that block is skipped. For instance:

```
x = 5
if x > 0:
    print("Positive")
```

Here, `"Positive"` will be printed only if `x > 0`. The Python reference defines the `if` statement grammar as:

```
if <expr>: <suite>
```

meaning the condition `<expr>` is followed by a colon and an indented suite. You can also write simple `if` statements on one line (for very short actions) or use boolean operators. For example, `if x or y:` runs if *either* `x` or `y` is true, whereas `if x and y:` requires *both* to be true.

`elif` and `else` Clauses

The `elif` clause lets you check additional conditions if the previous `if` (or `elif`) was false. You can have zero or more `elif` clauses. The final `else` block (if present) catches all cases not already handled. The structure is:

```
if <cond1>:
    ...
elif <cond2>:
    ...
elif <cond3>:
    ...
else:
    ...
```

Python tests `<cond1>`, then `<cond2>`, etc., and executes the first matching block. If none match and there is an `else`, that block runs. Only one block executes in the whole chain. For example:

```
score = 85
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
else:
    grade = 'F'
print(f"Grade = {grade}")
```

In this code, only the `elif score >= 80:` block will run, assigning `'B'`. As the Python tutorial notes, `elif` is short for "else if" and helps avoid deep nesting of `if` statements. You can use as many `elif` clauses as needed, but only one final `else` (if any) which must come last. If you omit the `else`, and no conditions are true, nothing happens.

Indentation and Code Blocks

In Python, **indentation is part of the syntax**. The code under an `if`, `elif`, or `else` must be indented consistently (typically 4 spaces) to form the block that executes when the condition is true. For example:

```
if x > 0:
    print("Positive")
    x = x - 1
else:
    print("Non-positive")
```

Here, the two lines under `if` are indented to the same level and both belong to the `if` block. The `else:` is not indented (it lines up with the `if`), and its indented lines form the `else` block. Python uses this **off-side rule**: contiguous statements at the same indent level form a group, or *suite*. If the indentation is incorrect or inconsistent, Python will raise an `IndentationError` (e.g. missing indent after `if:` or mismatched indents).

Indentation also resolves the "dangling else" problem: each `else` aligns with the closest preceding `if` at the same indent, so the structure is unambiguous. The Python style guide (PEP 8) recommends using 4 spaces per indentation level (no tabs) for readability. Always indent a suite (the block after `if:` or `elif:`) by the same amount. For instance:

```
if condition:
    statement1
    statement2
# Correct: both statements are part of the if-block

if condition:
statement1  # ✕ Error: this line must be indented
```

Proper indentation makes it clear which code is controlled by each conditional and avoids logic or syntax errors.

Common Use Cases

Python `if`/`elif`/`else` statements are used whenever you need decision-making or branching logic. Common examples include:

- **Input validation:** Check user input or function arguments before proceeding.
- `age = int(input("Age: "))`
- `if age < 0:`
- `    print("Invalid age!")`
- `elif age < 18:`
- `    print("Minor")`
- `else:`
- `    print("Adult")`
- **State-based logic:** Perform different actions depending on program state or mode (e.g. command menus, error handling).
- **Data classification:** Map values to categories, such as converting a numeric score to a grade (as above) or assigning labels based on ranges.
- **Flags and feature toggles:** Enable or disable functionality based on configuration or environment variables.

In each case, you check one condition (or chain of conditions) and branch accordingly. Only the branch whose condition is true runs.

Tips for Clear, Maintainable Conditionals

- **Keep conditions simple.** Use descriptive variable names and break complex expressions into well-named sub-conditions if needed. Instead of `if (x > 0 and x < 10 and y == z)`, consider separate checks or helper functions for readability.
- **Avoid deep nesting.** Excessive nested `if` statements (many levels of indent) can be hard to follow. You can often "invert" a condition and return early (a *guard clause*) or use `elif` to flatten logic. Real Python advises that very lengthy `if`/`elif` chains can be inelegant, and suggests looking for more Pythonic structures when possible. For example, if you're comparing one variable against many values, consider a dictionary mapping or (in Python 3.10+) a `match`/`case` statement as an alternative.
- **Use `elif` instead of multiple nested `if`s.** If you have mutually exclusive conditions, chaining with `elif` is clearer and prevents unnecessary checks. Remember that once a true condition is found, the rest are skipped (short-circuit behavior).
- **Be careful with logical operators.** Understand `and`, `or`, and `in`. For instance, `if x or y:` is true if *either* operand is true, while `if x and y:` requires both. Use parentheses to group complex expressions for clarity.

- **Follow style conventions.** As mentioned, use 4-space indents and align your `elif`/`else` with the original `if`. Keep lines under ~79 characters if possible. Use blank lines to separate large blocks. These practices (PEP 8) improve readability.
- **Comment non-obvious logic.** If a condition or branch is not immediately clear, add a comment or docstring. For example: `if age < 0: # negative ages are invalid`. This helps future readers (or your future self) understand the intent.
- **Test edge cases.** Ensure your conditions cover all relevant cases (e.g. equality vs. inequality boundaries) and that the `else` truly is a fallback for "all other" cases.