Python Control Flow: `if`, `elif`, `else`

Introduction

Control flow in Python allows a program to execute different blocks of code based on conditions. The primary control flow statements in Python are `if`, `elif`, and `else`. These statements enable decision-making and conditional execution in a program.

The `if` Statement

The `if` statement evaluates a condition, and if it is `True`, the corresponding block of code executes.

Syntax:

```
if condition:
    # Block of code
```

In [1]:
```
x = 10
if x > 5:
    print("x is greater than 5")
```

```
x is greater than 5
```

Explanation:

- The condition `x > 5` is evaluated.
- Since `10 > 5` is `True`, the indented block executes.

The `else` Statement

The `else` statement provides an alternative block that runs when the `if` condition is `False`.

Syntax:

```
if condition:
    # Block of code if condition is True
else:
    # Block of code if condition is False
```

In [2]:
```
x = 3
if x > 5:
    print("x is greater than 5")
else:
    print("x is 5 or less")
```

```
x is 5 or less
```

Explanation:

- The condition `x > 5` evaluates to `False`.
- The `else` block executes instead.

The `elif` (Else If) Statement

The `elif` statement allows checking multiple conditions sequentially. It prevents the need for multiple `if` statements.

Syntax:

```
if condition1:
    # Block of code if condition1 is True
elif condition2:
    # Block of code if condition2 is True
else:
    # Block of code if none of the above conditions are True
```

In [3]:
```
x = 7
if x > 10:
    print("x is greater than 10")
elif x > 5:
    print("x is between 6 and 10")
else:
    print("x is 5 or less")
```

```
x is between 6 and 10
```

Explanation:

- The first condition `x > 10` is `False`.
- The second condition `x > 5` is `True`, so its block executes.
- The `else` block does not execute since an earlier condition was met.

Nested `if` Statements

An `if` statement can be nested inside another `if`, `elif`, or `else` block to create more complex decision-making structures.

In [4]:
```
x = 15
if x > 10:
    print("x is greater than 10")
    if x > 20:
        print("x is also greater than 20")
    else:
        print("x is between 10 and 20")
```

```
x is greater than 10
x is between 10 and 20
```

Explanation:

- The first `if` condition (`x > 10`) is `True` , so it enters the block.
- Inside, another `if` condition checks if `x > 20` , which is `False` .
- The `else` block executes as a result.

Using Logical Operators

Logical operators (`and` , `or` , `not`) can combine multiple conditions.

In [6]:
```
x = 8
y = 3
if x > 5 and y < 5:
    print("Both conditions are True")
```

```
Both conditions are True
```

The Ternary (Conditional) Operator

Python allows a shorthand for `if-else` using the ternary operator.

Syntax:

```
value_if_true if condition else value_if_false
```

In [7]:
```
x = 10
y = "Greater" if x > 5 else "Smaller"
print(y)
```

```
Greater
```

Summary

- `if` checks a condition and executes its block if `True` .
- `else` runs when the `if` condition is `False` .
- `elif` allows multiple conditional checks.
- Nested `if` statements enable complex logic.
- Logical operators (`and` , `or` , `not`) combine conditions.
- The ternary operator provides a compact `if-else` expression.

These control flow statements are essential for decision-making in Python programs, allowing for flexible and dynamic code execution.