Python Control Flow Statements: The else Clauses on Loops

In Python, the `else` clause can be used with loops (`for` and `while`). This may be surprising at first since most people associate `else` with `if` statements. However, in loops, the `else` clause has a unique behavior:

- The `else` block is executed **only if the loop completes all its iterations without encountering a `break` statement**.
- If the loop is exited early because of a `break`, the `else` block is skipped.

The `else` clause in loops (`for` and `while`) in Python is a bit unusual because most people associate `else` with `if` statements. In the context of loops, the `else` clause is executed only when the loop finishes normally, meaning it wasn't interrupted by a `break` statement.

Syntax for Python `else` Clause in Loops

The `else` clause can be used with both `for` and `while` loops in Python. Here's the general syntax:

For Loop with else

```
for item in iterable:
    # Code block to execute for each item
    if condition:
        break  # Exit the loop early
else:
    # Code block to execute if the loop completes without a break
```

While Loop with else

```
while condition:
    # Code block to execute while the condition is True
    if condition_to_break:
        break  # Exit the loop early
else:
    # Code block to execute if the loop completes without a break
```

How It Works:

1. **With a `for` loop**:
    - The `else` block runs if the loop completes all its iterations without hitting a `break`.
    - If the loop is terminated by a `break`, the `else` block is skipped.
2. **With a `while` loop**:
    - The `else` block runs if the `while` loop condition becomes `False` naturally.
    - If the loop is terminated by a `break`, the `else` block is skipped.

Example with a `for` loop

Let's say we're searching for a specific number in a list.

```
# List of numbers
numbers = [1, 2, 3, 4, 5]

# Number we want to find
target = 6

# Iterate over the list
for num in numbers:
    if num == target:
        print("Found the target!")
        break
else:
    print("Target not found in the list.")
```

Explanation:

- The loop checks each number to see if it matches the `target`.
- If the number is found, the loop breaks, and the `else` clause is skipped.
- If the loop finishes without finding the target (i.e., without a `break`), the `else` block runs, printing "Target not found in the list."

3. Combined Example

Use Case: Login system with limited attempts.

```
max_attempts = 3
correct_password = "secret123"

for attempt in range(1, max_attempts + 1):
    password = input(f"Attempt {attempt}: Enter password: ")
    if password != correct_password:
        print("Wrong password. Try again.")
        continue  # Skip to next attempt
    else:
        print("Login successful!")
        break  # Exit loop on success
else:
    print("Account locked. Too many failed attempts.")
```

Output (if user fails 3 times):

```
Attempt 1: Enter password: hello
Wrong password. Try again.
Attempt 2: Enter password: test
Wrong password. Try again.
Attempt 3: Enter password: 123
Wrong password. Try again.
Account locked. Too many failed attempts.
```

Example with a `while` loop

```
# Counter
i = 1

# Loop condition
while i <= 5:
    if i == 3:
        print("Breaking the loop")
        break
    print(i)
    i += 1
else:
    print("Loop finished without breaking.")
```

Explanation:

- The loop runs while `i` is less than or equal to 5.
- If `i` equals 3, the loop breaks.
- Since the loop is broken before it naturally ends, the `else` block is skipped.

Why Use the `else` Clause with Loops?

Using `else` with loops can be helpful when you're performing a search or some operation where you want to know if the loop completed successfully or was interrupted by a `break`. It's a clean way to handle scenarios where the loop might end early.

Example #: for..else

```
for x in range(3):
    print(x)
else:
    print('Final x = %d' % (x))
```