List Comprehensions: Python's Shortcut Magic!

Hey Python adventurers! Today, we're learning **list comprehensions**—a fast, cool way to build lists from loops in one line. Think of it as a superpower for creating lists without writing full `for` loops. We'll:

- Compare loops vs. comprehensions.
- Add conditions to filter.
- Try fun examples!

Run the cells, tweak the code, and tackle the challenges—let's shrink some loops!

What Are List Comprehensions?

A **list comprehension** creates a list from an iterable (like a range or list) in one line. Instead of:

```
new_list = []
for item in some_iterable:
    new_list.append(something)
```

We write:

```
new_list = [something for item in some_iterable]
```

Python - Basic List Comprehension

Example 1: Traditional Loop vs. Comprehension

Goal: List of doubled numbers from 0 to 4

```
# Traditional way
doubled = []
for num in range(5):
    doubled.append(num * 2)
print("Traditional:", doubled)
```

```
Traditional: [0, 2, 4, 6, 8]
```

```
# Comprehension way
print([num * 2 for num in range(5)])
# print("Comprehension:", doubled_comp)
```

```
[0, 2, 4, 6, 8]
```

```
# Challenge: Triple it!
tripled = [n * 3 for n in range(4)]  # Fill in the range!
print("Tripled (0-3):", tripled)
# Try range(6) or multiply by 5!
```

Adding Conditions: Filter Your List

You can add an `if` inside a comprehension to pick only certain items—like a bouncer at a club!
Syntax: `[expression for item in iterable if condition]`

```
# Example 2: Only odd numbers tripled
# Traditional way
odds = []
for num in range(6):
    if num % 2 != 0:
        odds.append(num * 3)
print("Traditional odds:", odds)
```

```
Traditional odds: [3, 9, 15]
```

```
# Comprehension way
odds_comp = [num * 3 for num in range(6) if num % 2 != 0]
print("Comprehension odds:", odds_comp)  # [3, 9, 15]
```

```
# Challenge: Pick evens!
evens = [n * 2 for n in range(7) if n % 2 == 0]
print("Even doubles (0-6):", evens)
```

Using Lists as Input

Comprehensions aren't just for ranges—you can use any list! Let's transform some words.

```
# Example 3: Make words shouty
colors = ["red", "blue", "green"]
# Traditional way
shouty = []
for color in colors:
    shouty.append(color.upper())
print("Traditional shouty:", shouty)
```

```
Traditional shouty: ['RED', 'BLUE', 'GREEN']
```

```
# Comprehension way
shouty_comp = [color.upper() for color in colors]
print("Comprehension shouty:", shouty_comp)
```

```
# Challenge: Add "y" to animals!
animals = ["cat", "dog", "bird"]
y_animals = [animal + "y" for animal in animals]
print("Y animals:", y_animals)
```

Nested Comprehensions: Loops Inside Loops

You can nest comprehensions—like loops inside loops—but let's keep it simple for now!

```
# Example 4: Pairs of letters and numbers
# Traditional way
pairs = []
for letter in ["a", "b"]:
    for num in [1, 2]:
        pairs.append(letter + str(num))
print("Traditional pairs:", pairs)

```

```
# Comprehension way
pairs_comp = [letter + str(num) for letter in ["a", "b"] for num in [1, 2]]
print("Comprehension pairs:", pairs_comp)  # ["a1", "a2", "b1", "b2"]

```

```
# Challenge: Mix it!
mix = [x + str(y) for x in ["x", "y"] for y in [3, 4]]
print("Mixed pairs:", mix)
# Try ["p", "q"] with [5, 6]!
```

Wrap-Up: Comprehension Champs!

You've learned:

- **Basics**: `[expression for item in iterable]`.
- **Conditions**: Add `if` to filter.

- **Lists**: Transform any iterable in one line.

Keep practicing! Make a list of lowercase names, filter big numbers, or nest some pairs. What’s your favorite comprehension? Share it!