List comprehensions in Python provide a concise and expressive way to create new lists by applying an expression to each item in an iterable (like a list, range, or set) while optionally filtering elements. They are often used as a more readable and compact alternative to traditional loops for creating lists.

**Syntax**

new\_list = [expression for item in iterable if condition]

* **expression**: Defines what each item in the new list will look like.
* **item**: The variable representing the current element of the iterable.
* **iterable**: The source sequence or iterable.
* **condition** *(optional)*: A filter applied to include only certain items.

**Examples**

**1. Basic List Comprehension**

Creating a list of squares:

squares = [x\*\*2 for x in range(10)]

print(squares)

# Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

**2. Using a Condition**

Filtering for even numbers:

evens = [x for x in range(10) if x % 2 == 0]

print(evens)

# Output: [0, 2, 4, 6, 8]

**3. Nested Loops**

Generating a Cartesian product:

pairs = [(x, y) for x in range(3) for y in range(2)]

print(pairs)

# Output: [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)]

**4. With Functions**

Applying a function to each element:

def square(x):

    return x\*\*2

squared\_values = [square(x) for x in range(5)]

print(squared\_values)

# Output: [0, 1, 4, 9, 16]

**5. Conditional Expression**

Including an else clause:

labels = ["Even" if x % 2 == 0 else "Odd" for x in range(5)]

print(labels)

# Output: ['Even', 'Odd', 'Even', 'Odd', 'Even']

**6. Flattening a Nested List**

Flattening a 2D list:

matrix = [[1, 2], [3, 4], [5, 6]]

flat\_list = [item for sublist in matrix for item in sublist]

print(flat\_list)

# Output: [1, 2, 3, 4, 5, 6]

**Advantages of List Comprehensions**

* **Readability**: Compact and more expressive than traditional loops.
* **Performance**: Often faster than equivalent for-loops due to optimization.

**Exercises Beginner Level**

1. **Squares of Numbers** Generate a list of squares for numbers from 1 to 10.

# Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

1. **Even Numbers** Create a list of even numbers between 1 and 20.

# Output: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

1. **First Letter of Words** Extract the first letter from each word in a list of strings.

words = ["apple", "banana", "cherry", "date"]

# Output: ['a', 'b', 'c', 'd']

1. **Filter Multiples of 3** From a list of numbers, filter only those that are multiples of 3.

numbers = [3, 5, 9, 12, 15, 20, 25, 30]

# Output: [3, 9, 12, 15, 30]

**Intermediate Level**

1. **Cartesian Product** Create a list of all pairs (x, y) where x is from [1, 2, 3] and y is from [4, 5, 6].

# Output: [(1, 4), (1, 5), (1, 6), (2, 4), ..., (3, 6)]

1. **Flatten a Nested List** Convert a 2D list into a 1D list.

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

1. **Conditional Transformation** Replace numbers in a list: if even, keep the number; if odd, replace it with -1.

numbers = [1, 2, 3, 4, 5]

# Output: [-1, 2, -1, 4, -1]

1. **Reverse Strings** Reverse each string in a list.

strings = ["hello", "world", "python"]

# Output: ['olleh', 'dlrow', 'nohtyp']

**Advanced Level**

1. **Find Common Elements** Create a list of common elements from two lists using list comprehension.

list1 = [1, 2, 3, 4, 5]

list2 = [3, 4, 5, 6, 7]

# Output: [3, 4, 5]

1. **Prime Numbers** Generate a list of prime numbers between 2 and 50.

# Output: [2, 3, 5, 7, 11, 13, ..., 47]

1. **Word Length Filter** From a list of words, create a new list of words that have more than 3 letters.

words = ["hi", "hello", "to", "world", "a", "python"]

# Output: ['hello', 'world', 'python']

1. **Nested List Comprehension** Create a multiplication table (1 to 5) as a nested list.

# Output: [[1, 2, 3, 4, 5], [2, 4, 6, 8, 10], ..., [5, 10, 15, 20, 25]]