Understanding List Comprehension in Python

MUKESH KUMAR

AGENDA

- What is List Comprehension?
- Basic Syntax
- Using Conditional Statements
- Advantages of List Comprehension
- Cautionary Notes
- Examples and Exercises
- Wrap-Up and Best Practices

What is List Comprehension?

- **Definition**: List comprehension provides a concise way to create lists.
- **Key benefits**: Readability, Efficiency.

Example: `[x**2 for x in range(5)]` -> `[0, 1, 4, 9, 16]`

Basic Syntax

```
[expression for item in iterable]
```

- **expression**: What you want to include in the new list (modified item).
- **item**: The variable representing each element in the iterable.
- **iterable**: The sequence you're iterating over (list, tuple, string, range, etc.).
- **if condition** (optional): Filters elements based on a boolean condition.

Example

```
# Using list comprehension to create a list of squares of numbers
numbers = [1, 2, 3, 4, 5]

# Basic list comprehension
squares = [x**2 for x in numbers]

print(f"Squares of numbers: {squares}")
```

- Output:

```
Squares of numbers: [1, 4, 9, 16, 25]
```

Basic Syntax Practice

- Create a list containing the cubes of numbers from 1 to 10.
- Convert a list of strings to lowercase.
- Given a list of numbers, create a new list with each number multiplied by its index.
- Generate a list of tuples, where each tuple contains a number and its square
- Given two lists, create a new list by concatenating corresponding elements as strings

Using Conditional Statements

- **Syntax with if:**

```
[expression for item in iterable if condition]
```

- **expression**: What you want to include in the new list (modified item).
- **item**: The variable representing each element in the iterable.
- **iterable**: The sequence you're iterating over (list, tuple, string, range, etc.).
- **if condition** (optional): Filters elements based on a boolean condition.

Example

```python
# Using list comprehension with an if condition to filter even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# List comprehension with if condition to filter even numbers
even_numbers = [x for x in numbers if x % 2 == 0]

print(f"Even numbers: {even_numbers}")
```

- Output:

```
Even numbers: [2, 4, 6, 8, 10]
```

If Condition Practice

- Create a list of numbers greater than 10 from a given list
- Filter a list of strings to include only those that contain the letter "a".
- Extract the even numbers from a list of mixed data types (including strings, etc.)
- Given a list of words, filter out those that start with a vowel
- From a list of integers, keep only the positive numbers that are also divisible by 3

ENUMERATE & ZIP

enumerate()

- The enumerate() function adds a counter to an iterable and returns it as an enumerate object.
- This is helpful when you need both the index and the value of items in a loop.

Enumerate

- Enumerate code:

```
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits):
    print(index, fruit)
```

- Code:

```
0 apple
1 banana
2 cherry
```

Zip

- The zip() function combines two or more iterables (like lists or tuples) element-wise into tuples.
- It stops when the shortest iterable is exhausted.

Zip

- Zip Code:

```
names = ['Alice', 'Bob', 'Charlie']
scores = [85, 92, 88]
zipped = zip(names, scores)

for name, score in zipped:
    print(name, score)
```

- Output:

```
Alice 85
Bob 92
Charlie 88
```

Key Differences:

- **enumerate()** is used to iterate through an iterable while keeping track of the index.
- **zip()** is used to combine multiple iterables element-wise.

Using Conditional Statements

- **Syntax with if-else:**

```
[expression1 if condition else expression2 for item in iterable]
```

- **expression1:** This is the expression that will be evaluated and included in the new list *if* the condition evaluates to True.
- **else expression2:** This is the expression that will be evaluated and included in the new list *if* the condition evaluates to False.

Example

```python
# Using list comprehension with if and else to label even and odd numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# List comprehension with if and else to label even and odd
labels = ['Even' if x % 2 == 0 else 'Odd' for x in numbers]

print(f"Labels for numbers: {labels}")
```

- Output:

```
['Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even']
```

Practice

- Given a list of numbers, create a new list where each number is replaced by "positive" if it's positive, "negative" if it's negative, and "zero" if it's zero.
- Given a list of words, convert them to uppercase if they are shorter than 5 characters, and to lowercase otherwise.
- Create a list where even numbers are doubled, and odd numbers are tripled.
- Given a list of strings, replace each string with its length if the length is even, and with the string itself if the length is odd.
- From a list of numbers, create a new list where each number is replaced by its square if it's positive and its cube if it's negative.
- Given a list of items with prices, apply a 10% discount if the price is greater than $50, and no discount otherwise.

Advantages

- Concise and elegant code.
- Improves readability and reduces lines of code.
- Fast execution compared to traditional loops.

Caution

- Overuse can lead to less readability for complex comprehensions.
- Avoid nested comprehensions for highly complex logic.

Wrap-Up

- **Summary:** List comprehensions are a powerful tool for creating lists concisely and efficiently.
- **Encourage best practices:** Keep them simple and readable.

List Comprehension with all iterables

- with Lists
- with Tuples
- with Sets
- with Strings
- with range()
- with Dictionaries (Keys & Values)
- with enumerate()
- with zip()

List Comprehension with Lists

```
numbers = [1, 2, 3, 4, 5]
squared = [x**2 for x in numbers]
print(squared)  # Output: [1, 4, 9, 16, 25]
```

List Comprehension with Tuples

```
numbers_tuple = (1, 2, 3, 4, 5)
cubed = [x**3 for x in numbers_tuple]
print(cubed)  # Output: [1, 8, 27, 64, 125]
```

List Comprehension with Sets

```
numbers_set = {1, 2, 3, 4, 5}
double_values = [x * 2 for x in numbers_set]
print(double_values)  # Output: [2, 4, 6, 8, 10]
```

List Comprehension with Strings

```
text = "hello"
uppercase_chars = [char.upper() for char in text]
print(uppercase_chars)  # Output: ['H', 'E', 'L', 'L', 'O']
```

List Comprehension with range()

```python
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers)  # Output: [0, 2, 4, 6, 8]
```

List Comprehension with Dictionaries (Keys & Values)

- **Extracting Keys:**

```
my_dict = {'a': 10, 'b': 20, 'c': 30}
keys_list = [key for key in my_dict]
print(keys_list)  # Output: ['a', 'b', 'c']
```

- **Extracting Values:**

```
values_list = [value for value in my_dict.values()]
print(values_list)  # Output: [10, 20, 30]
```

- Extracting Key-Value Pairs as Tuples

```
key_value_pairs = [(key, value) for key, value in my_dict.items()]
print(key_value_pairs)  # Output: [('a', 10), ('b', 20), ('c', 30)]
```

List Comprehension with enumerate()

```
items = ['a', 'b', 'c']
indexed_items = [(index, item) for index, item in enumerate(items)]
print(indexed_items)  # Output: [(0, 'a'), (1, 'b'), (2, 'c')]
```

List Comprehension with zip()

```python
names = ["Alice", "Bob", "Charlie"]
scores = [85, 90, 78]
combined = [(name, score) for name, score in zip(names, scores)]
print(combined)  # Output: [('Alice', 85), ('Bob', 90), ('Charlie', 78)]
```