List Comprehensions

List comprehensions in Python provide a concise way to create lists by applying an expression to each item in an existing iterable (such as a list, tuple, or range) and filtering the items based on certain conditions.

The general syntax of a list comprehension is:

[expression for item in iterable if condition]

Here's what each part means:

- **`expression`**: This is the operation or calculation you want to perform on each item.

- **`item`**: This represents each element in the iterable.

- **`iterable`**: This is the existing collection of items (e.g., a list, tuple, or range) that you want to process.

- **`condition`** (optional): This allows you to include only the items that meet a certain criteria.

Let's go through some examples with their outputs:

Example 1: Creating a list of squares

```
squares = [x ** 2 for x in range(5)]
print(squares)
```

Output:

```
[0, 1, 4, 9, 16]
```

Example 2: Filtering even numbers

```
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers)
```

Output:

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

Example 3: Creating a list of tuples

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

Output:

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

Example 4: Filtering using if-else

```
results = ['Pass' if score >= 60 else 'Fail' for score in [75, 30, 85, 50]]
print(results)
```

Output:

```
['Pass', 'Fail', 'Pass', 'Fail']
```

Example 5: Creating a list of strings

```
names = ['John', 'Jane', 'Jim']
name_lengths = [len(name) for name in names]
print(name_lengths)
```

Output:

```
[4, 4, 3]
```

These examples demonstrate how list comprehensions can be used to perform operations on elements of an iterable and optionally filter them based on certain conditions. They are a powerful and concise feature of Python that can make code more readable and efficient.