Mastering List Comprehensions And Expressions In Python

Craig Godden-Payne
May 13 · 2 min read ★

Photo by Glenn Carstens-Peters on Unsplash

> *List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member*

> *of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.*

List comprehension can be used in place of for loops, when working with lists. There is a basic syntax when working with a comprehension, and it looks something like this:

```
[ expression for x in list if conditional ]
```

Iterating through a string

In this example, we are going to iterate through a string, and append each letter into a list. Then we will print the list

- Using a for loop

```
letters = []
for letter in 'dinosaur':
    letters.append(letter)

print(letters)

> ['d', 'i', 'n', 'o', 's', 'a', 'u', 'r']
```

- Using list comprehension

```
letters = [ letter for letter in 'dinosaur' ]
print(letters)

> ['d', 'i', 'n', 'o', 's', 'a', 'u', 'r']
```

- Using a lambda expression

```
letters = list(map(lambda letter: letter, 'dinosaur'))
print(letters)

> ['d', 'i', 'n', 'o', 's', 'a', 'u', 'r']
```

Working with conditionals

You can use list comprehensions and utilise conditional statements to modify an existing list

Here, we use the range function to return a list of numbers, from 0 to 20, then using a conditional, we modulus by 2 == 0 to only return even numbers.

```
numbers = [ number for number in range(20) if number % 2 == 0 ]
print(numbers)

> [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

You can nest conditionals, like in the next example. We use the same range, but this time we only want even numbers and numbers not in 1, 2, 3, 4, 5

```
numbers = [ num for num in range(20) if num not in [1,2,3,4,5] if
num % 2 == 0 ]
print(numbers)

> [0, 6, 8, 10, 12, 14, 16, 18]
```

You can use if else as well with conditionals. Again using the same range, we output whether each number is even or odd.

```
evenodd = [ 'Even' if number % 2 == 0 else 'Odd' for number in
range(20) ]
print(evenodd)

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

Working with nested loops

List comprehension can be used with nested loops. Using a traditional *for in* loop, we would iterate something like this:

```
nested_list = [[1,2,3],[4,5,6]]
flattened_list = []
for x in nested_list:
```

```
    for y in x:
        flattened_list.append(y)

print(flattened_list)

> [1, 2, 3, 4, 5, 6]
```

To convert this to use list comprehension, we need to do the following.

```
nested_list = [[1,2,3],[4,5,6]]
flattened_list = [ y for x in nested_list for y in x ]
print(flattened_list)

> [1, 2, 3, 4, 5, 6]
```


Python