Python List Comprehension

List comprehension offers a concise way to create a new list based on the values of an existing list.

Suppose we have a list of numbers and we desire to create a new list containing the double value of each element in the list.

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

# list comprehension to create new list
doubled_numbers = [num * 2 for num in numbers]


print(doubled_numbers)
```

Output

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

Here is how the list comprehension works:

Python List Comprehension

Syntax of List Comprehension

```
[expression for item in list if condition == True]
```

Here,

for every `item` in `list`, execute the `expression` if the `condition` is `True`.

Note: The `if` statement in list comprehension is optional.

for Loop vs. List Comprehension

List comprehension makes the code cleaner and more concise than `for` loop.

Let's write a program to print the square of each list element using both `for` loop and list comprehension.

for Loop

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

# for loop to square each elements
for num in numbers:
    square_numbers.append(num * num)


print(square_numbers)

# Output: [1, 4, 9, 16, 25]
```

List Comprehension

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

# create a new list using list comprehension
square_numbers = [num * num for num in numbers]


print(square_numbers)

# Output: [1, 4, 9, 16, 25]
```

It's much easier to understand list comprehension once you know Python for loop().

Conditionals in List Comprehension

List comprehensions can utilize conditional statements like ifâ€¦else to filter existing lists.

Let's see an example of an `if` statement with list comprehension.

```
# filtering even numbers from a list
even_numbers = [num for num in range(1, 10) if num % 2 == 0 ]


print(even_numbers)

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

Here, list comprehension checks if the number from `range(1, 10)` is even or odd. If even, it appends the number in the list.

Note: The `range()` function generates a sequence of numbers. To learn more, visit Python range().

if...else With List Comprehension

Let's use `if...else` with list comprehension to find even and odd numbers.

```
numbers = [1, 2, 3, 4, 5, 6]

# find even and odd numbers
even_odd_list = ["Even" if i % 2 == 0 else "Odd" for i in numbers]


print(even_odd_list)
```

Output

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

Here, if an item in the *numbers* list is divisible by **2**, it appends `Even` to the list *even_odd_list*. Else, it appends `Odd`.

Nested if With List Comprehension

Let's use nested `if` with list comprehension to find even numbers that are divisible by **5**.

```
# find even numbers that are divisible by 5
num_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]


print(num_list)
```

Output

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Here, list comprehension checks two conditions:

1. if *y* is divisible by **2** or not.
2. if yes, is *y* divisible by **5** or not.

If *y* satisfies both conditions, the number appends to *num_list*.

Example: List Comprehension with String

We can also use list comprehension with iterables other than lists.

```
word = "Python"
vowels = "aeiou"

# find vowel in the string "Python"
result = [char for char in word if char in vowels]


print(result)

# Output: ['o']
```

Here, we used list comprehension to find vowels in the string `'Python'`.

More on Python List Comprehension

Nested List Comprehension

We can also use nested loops in list comprehension. Let's write code to compute a multiplication table.

```
multiplication = [[i * j for j in range(1, 6)] for i in range(2, 5)]

print(multiplication)
```

Output

```
[[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]
```

Here is how the nested list comprehension works:

Nested Loop in List Comprehension

Let's see the equivalent code using nested `for` loop.

Equivalent Nested for Loop

```
multiplication = []

for i in range(2, 5):
    row = []
    for j in range(1, 6):
        row.append(i * j)
    multiplication.append(row)

print(multiplication)
```

Here, the nested `for` loop generates the same output as the nested list comprehension. We can see that the code with list comprehension is much cleaner and concise.

List Comprehensions vs Lambda Functions

Along with list comprehensions, we also use lambda functions to work with lists.

While list comprehension is commonly used for filtering a list based on some conditions, lambda functions are commonly used with functions like `map()` and `filter()`.

They are used for complex operations or when an anonymous function is required.

Let's look at an example.

```
numbers = [5, 6, 7, 8, 9]

# create a new list using a lambda function
square_numbers = list(map(lambda num : num**2 , numbers))


print(square_numbers)
```

Output

```
[25, 36, 49, 64, 81]
```

We can achieve the same result using list comprehension by:

```
# create a new list using list comprehension
square_numbers = [num ** 2 for num in numbers]
```

If we compare the two codes, list comprehension is straightforward and simpler to read and understand.

So unless we need to perform complex operations, we can stick to list comprehension.

Visit Python Lambda/ Function to learn more about the use of lambda functions in Python.

Table of Contents