What are list comprehensions?

List comprehensions are a tool for transforming one list (any iterable actually) into another list. During this transformation, elements can be conditionally included in the new list and each element can be transformed as needed.

List comprehensions in Python are great, but mastering them can be tricky because they don't solve a new problem: they just provide a new syntax to solve an existing problem.

From loops to comprehensions

Every list comprehension can be rewritten as a `for` loop but not every `for` loop can be rewritten as a list comprehension .The key to understanding when to use list comprehensions is to practice identifying problems that smell like list comprehensions. If you can rewrite your code to look just like this `for` loop, you can also rewrite it as a list comprehension:

Let us take an example of getting the squares of the numbers in a list

```
# The list of numbers
list_of_numbers = [1, 2, 4, 6, 11, 14, 17, 20]
```

The usual way to do this using `for` loop is described below:

```
# Let us first initialize a list where we will be appending the squares in each iteration

squared_numbers = []

for number in list_of_numbers:
  # Use the append method to add the numbers one by one to our list
  squared_numbers.append(number**2)
  # print(squared_numbers)

print(f"The list of squared numbers is {squared_numbers}")
```

```
The list of squared numbers is [1, 4, 16, 36, 121, 196, 289, 400]
```

We can do the same task using a list comprehension

```
# Getting the list of squares using list comprehension
squared_numbers = [number**2 for number in list_of_numbers]
print(squared_numbers)
```

```
[1, 4, 16, 36, 121, 196, 289, 400]
```

Further we can go ahead and use some conditionals.

Suppose we wanted to get the squares of only those numbers which are even

```
print(list_of_numbers)
```

```
[1, 2, 4, 6, 11, 14, 17, 20]
```

```
# Writing list comprehension using conditionals
even_squared_numbers = [number**2 for number in list_of_numbers if number%2==0]
```

```
print(even_squared_numbers)
```

```
[4, 16, 36, 196, 400]
```

```
# Writing list comprehension using conditionals
odd_squared_numbers = [number**2 for number in list_of_numbers if number%2!=0]
print(odd_squared_numbers)
```

```
[1, 121, 289]
```

```
new_numbers = [number**2 if number%2!=0 else number**3 for number in list_of_numbers]
print(new_numbers)
```

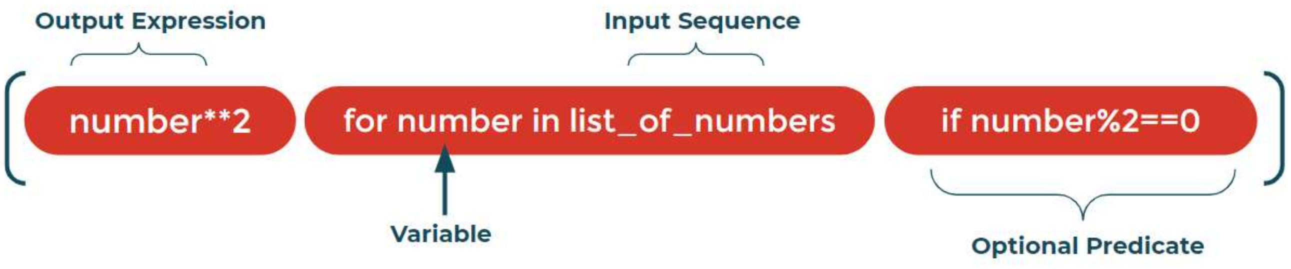
```
[1, 8, 64, 216, 121, 2744, 289, 8000]
```

```
# Writing list comprehension using conditionals
odd_squared_numbers = [number**2 if number%2!=0 else number**3 for number in list_of_numbers ]
print(odd_squared_numbers)
```

```
[1, 8, 64, 216, 121, 2744, 289, 8000]
```

A list comprehension consists of the following parts:

- An Input Sequence.
- A Variable representing members of the input sequence.
- An Optional Predicate expression.
- An Output Expression producing elements of the output list from members of the Input Sequence that satisfy the predicate.



- The iterator part iterates through each member `number` of the input sequence `list_of_numbers`.
- The predicate checks if the member is even.
- If the member is even then it is passed to the output expression, squared, to become a member of the output list.

Using `for` loop

```
even_squared_numbers = []

for number in list_of_numbers:
  if number%2 == 0:
    even_squared_numbers.append(number**2)
```

Using list comprehension

```
even_squared_numbers = [number**2 for number in list_of_numbers if number%2==0]
```

So how did we convert our `for` loop into a list comprehension?

- Copying the variable assignment for our new empty list
- Copying the expression `number**2` that we've been appending into this new list
- Copying the for loop line, excluding the final `:`
- Copying the if statement line, also without the `:`

Writing list comprehensions for nested loops

Let's go ahead and write a nested for loop

```
list(range(4))
```

```
[0, 1, 2, 3]
```

```
list_of_numbers = []

for num1 in range(4):
  for num2 in range(3):
    # print(num1,num2)
    list_of_numbers.append(num1)

print(list_of_numbers)
```

```
[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3]
```

```
# We can write it in a slightly different way
list_of_numbers = [num1 for num1 in range(4) for num2 in range(3)]
print(list_of_numbers)
```

```
[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3]
```

Note: My brain wants to write this list comprehension as:

```
list_of_numbers = [num2**2 for num2 in range(3) for num1 in range(4)]
print(list_of_numbers)
```

```
[0, 0, 0, 0, 1, 1, 1, 1, 4, 4, 4, 4]
```

But that's not right! I've mistakenly flipped the `for` loops here. The correct version is the one above. When working with nested loops in list comprehensions remember that the for clauses remain in the same order as in our original for loops.

Now let us see how we can create list of lists using list comprehension

```
sample = [[]]
len(sample)

sample[0]
```

```
[]
```

```
list_of_lists = []

for num1 in range(4):
  # Append an empty sublist inside the list
  list_of_lists.append([])  # [[0]] , list_of_lists[1]
  # print(list_of_lists)

  for num2 in range(3):
    # Append the elements within the sublist
    list_of_lists[num1].append(num2)  #list_of_lists[0]

print(list_of_lists)
```

```
[[0, 1, 2], [0, 1, 2], [0, 1, 2], [0, 1, 2]]
```

```
# Using list comprehension we can write the same thing
list_of_lists = [[num2 for num2 in range(3)] for num1 in range(4)]
print(list_of_lists)
```

```
[[0, 1, 2], [0, 1, 2], [0, 1, 2], [0, 1, 2]]
```

```
# Using list comprehension we can write the same thing
list_of_lists = [[num1 for num2 in range(3)] for num1 in range(4)]
print(list_of_lists)
```

```
[[0, 0, 0], [1, 1, 1], [2, 2, 2], [3, 3, 3]]
```

Up till now, we have only implemented list comprehension for list objects. We know that there are other iterables in Python which are sequence of elements such as strings, tuples etc. Let us try and apply list comprehension to iterate over their elements.

```
heisenberg_quote = "It ceases to exist without me. No, you clearly don't know who you're talking to, so let me clue you in. I am not in dang
```

```
words_by_walter = heisenberg_quote.split(' ')
print(words_by_walter)
```

```
['It', 'ceases', 'to', 'exist', 'without', 'me.', 'No,', 'you', 'clearly', "don't", 'know', 'who', "you're", 'talking', 'to,', 'so', 'le
```

```
heisenberg_quote.split('.')
```

```
['It ceases to exist without me',
 " No, you clearly don't know who you're talking to, so let me clue you in",
 ' I am not in danger, Skyler',
 ' I am the danger',
 '']
```

```
# Let u store the first letter of each word in another list

first_letters = []

for word in words_by_walter:
  first_letters.append(word[0])

# Print the first_letters list
print(first_letters)
```

```
['I', 'c', 't', 'e', 'w', 'm', 'N', 'y', 'c', 'd', 'k', 'w', 'y', 't', 't', 's', 'l', 'm', 'c', 'y', 'i', 'I', 'a', 'n', 'i', 'd', 'S',
```

```
# The same task but now with list comprehension
first_letters = [word[0] for word in words_by_walter]
print(first_letters)
```

```
['I', 'c', 't', 'e', 'w', 'm', 'N', 'y', 'c', 'd', 'k', 'w', 'y', 't', 't', 's', 'l', 'm', 'c', 'y', 'i', 'I', 'a', 'n', 'i', 'd', 'S',
```

```
heisenberg_quote.split(',')
```

```
['It ceases to exist without me. No',
 " you clearly don't know who you're talking to",
 ' so let me clue you in. I am not in danger',
 ' Skyler. I am the danger.']
```

```
my_string = 'Alphabet Inc.'

list_of_characters = [char for char in my_string]
print(list_of_characters)
```

```
['A', 'l', 'p', 'h', 'a', 'b', 'e', 't', ' ', 'I', 'n', 'c', '.']
```

We can also use list comprehension to get a tuple as well.

```
# Suppose we have a tuple of days
days = ('Monday' ,'Tuesday' ,'Wednesday' ,'Thursday' ,'Friday','Saturday','Sunday')
```

```
tuple(day for day in days)
```

```
('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')
```

```
# Let us create a tuple where we want to remove the word 'day' from each tuple element
days_without_day = tuple([day[:-3] for day in days])
print(days_without_day)
```

```
('Mon', 'Tues', 'Wednes', 'Thurs', 'Fri', 'Satur', 'Sun')
```

Note we cannot just specify parentheses and get this to work. It would create a generator object so we need to call the tuple function

Set Comprehension

```
# Using a for loop
first_letters = set()
for word in words_by_walter:
    first_letters.add(word[0])

print(first_letters)
```

```
{'N', 'i', 'e', 't', 'd', 'k', 'w', 'l', 'n', 'y', 'S', 's', 'I', 'c', 'm', 'a'}
```

```
# Using set comprehension
first_letters = {word[0] for word in words_by_walter}
print(first_letters)
```

```
{'N', 'i', 'e', 't', 'd', 'k', 'w', 'l', 'n', 'y', 'S', 's', 'I', 'c', 'm', 'a'}
```

Dictionary Comprehensions

```
country_city_list = [('India','New Delhi'),('Australia','Canberra'),('United States','Washington DC'),('England','London')]
```

```
# Lets convert the list of tuples to a dictionary
country_city_dict = dict(country_city_list)
print(country_city_dict)
```

```
{'India': 'New Delhi', 'Australia': 'Canberra', 'United States': 'Washington DC', 'England': 'London'}
```

Let us try to swap the keys and values for this dictionary

```
# The conventional for loop approach
flipped = {}
for country, city in country_city_dict.items():
    flipped[city] = country

print(flipped)
```

```
{'New Delhi': 'India', 'Canberra': 'Australia', 'Washington DC': 'United States', 'London': 'England'}
```

```
# The new dictionary comprehension
flipped = {city:country for country, city in country_city_dict.items()}
print(flipped)
```

```
{'New Delhi': 'India', 'Canberra': 'Australia', 'Washington DC': 'United States', 'London': 'England'}
```

```
# We can further add conditionals
flipped = { city : country for country, city in country_city_dict.items() if country[0] not in ['E','A']}
print(flipped)
```

```
{'New Delhi': 'India', 'Washington DC': 'United States'}
```

```
# The new dictionary comprehension
flipped = {country[:2] : city[-2:] for country, city in country_city_dict.items()}
print(flipped)
```

```
{'In': 'hi', 'Au': 'ra', 'Un': 'DC', 'En': 'on'}
```

```
print(country_city_dict)
```

```
{'India': 'New Delhi', 'Australia': 'Canberra', 'United States': 'Washington DC', 'England': 'London'}
```

Start coding or generate with AI.