How to use List comprehension in Python

You may have heard about Python's List Comprehension and may have used it without really understanding them. Therefore, this article will introduce and guide you how to use List Comprehension in Python.

You may have heard about List Comprehension in Python programming language and may have used it without really understanding them. Therefore, this article will introduce and guide you how to use List Comprehension in Python.

What is List Comprehension in Python?

List Comprehension (how to write short code to create a complex list) may sound complicated, but it is not. In Python, it is simply a quick way to filter or edit a list based on some criteria. List Comprehension helps you write short code (especially when in an existing loop) and easier to read and view code.

See also: Loop techniques in Python

How to use List Comprehensions in Python

Note : These examples all use Python 3.6.

Consider this code, copy an array (array) and turn each letter in this array into capital letters. It does this by repeating each item in the array.

```
letters = ['a', 'b', 'c', 'd'] print(letters) upper_letters = [] for letter in l
```

```
$python3 main.py
['a', 'b', 'c', 'd']
['A', 'B', 'C', 'D']
```

Now the following example is the same as above, but implemented in a single line of code using the basic List Comprehension Python:

```
letters = ['a', 'b', 'c', 'd'] print(letters) upper_letters = [x.upper() for x i
```

```
$python3 main.py
['a', 'b', 'c', 'd']
['A', 'B', 'C', 'D']
```

As you can see, the results of the above two codes are the same but the process uses more code when using List Comprehension.

If you find it difficult to understand, analyze the example using List Comprehension above.

This example creates a list, called **letters,** to store lowercase letters 'a', 'b', 'c' and 'd'. Suppose you want to turn all elements in this list into uppercase without using List Comprehension, you will have to create a new list of results (called **upper_letters**), loop through all elements in the list. book **letters** , convert each letter (and store it in **result**), and then add capital letters to the new list. You see, there are too many things to do.

List Comprehension here is almost equivalent, replacing the loop. It converts each letter in the **letters** list and returns the result as a new list. List Comprehension only works on lists and returns on new lists.

There are three parts in List Comprehension. List Comprehension must start and end with square brackets (**[** and **]**). This is how it is designed and allows Python to know that you will work with a list.

Within these brackets, you need to start with the result. This is what you want to do with each element in the list. In the above example, the following code converts each element (referenced by the variable name **x**) into uppercase using the **upper () method** , which is part of the Python base library:

```
[x.upper() # will not run, only half the comprehension at this point
```

Next, you need to tell Python the list to work on and assign each individual element with a variable. This is exactly the same as the loop in the previous example.

```
for x in letters
```

Every time the loop goes through the list, the value of **x** will change to any existing element. It will start with "a", and then "b", and so on.

If you put it all together (and assign it to a variable called **upper_letters**), you do the following:

```
upper_letters = [x.upper() for x in letters]
```

Now, **upper_letters** will contain a list of uppercase letters, starting with "A", and then "B", etc.

The third part of List comprehension in Python

As mentioned above, this is the third part of List Comprehension. Once you've done the two steps above, you can add optional conditions. This is like using an **if statement** to tell Python to create a new list, based on the old list, but only include elements that match the proposed criteria.

This is the implementation code:

```
ages = [1, 34, 5, 7, 3, 57, 356] print(ages) old_ages = [x for x in ages if x > 1
```

```
$python3 main.py
[1, 34, 5, 7, 3, 57, 356]
[34, 57, 356]
```

This example uses a new list called **ages** . The list of **old_ages** is created using List Comprehension. The **if-** end condition means that only elements in the list that match the new criteria are added to the new list. In this example, any age greater than 10 is added to the list.

In case you should not use List Comprehension

List Comprehension is great, but not all cases can use it. You should not use it when there is more than one condition.

```
old_ages = [x for x in ages if x > 10 and x  100 and x is not None]
```

This code still works but it starts long and confusing. Similarly, any code that has more than one simple function should not use List Comprehension. In this example, you will receive an error:

```
letters = ['a', 'b', 'c', 'd', 2] print(letters) upper_letters = [x.upper() for x
```

```
$python3 main.py
['a', 'b', 'c', 'd', 2]
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    upper_letters = [x.upper() for x in letters]
  File "main.py", line 4, in <listcomp>
    upper_letters = [x.upper() for x in letters]
AttributeError: 'int' object has no attribute 'upper'
```

This is perfectly valid code, but because you can't write a number, it doesn't work. In this case, you should use a loop because some exceptions can be made:

```
letters = ['a', 'b', 'c', 'd', 1] print(letters) upper_letters = [] for letter i
```

```
$python3 main.py
['a', 'b', 'c', 'd', 1]
['A', 'B', 'C', 'D']
```

Now you know how to use List Comprehension, so there is no reason not to use it to simplify the code, making it easier to understand.

I wish you all success!

See more:

1. More than 100 Python exercises have solutions (sample code)
2. How to install Python on Windows, macOS, Linux
3. Python data type: string, number, list, tuple, set and dictionary