Theory: List comprehension

28 minutes 10 / 16 problems solved

Start practicing

List comprehension is a way of making new lists. It allows you to create a list from any iterable object in a concise and efficient manner. See the basic syntax below:

```
# list comprehension syntax
new_list = [x for x in some_iterable]
```

Here you can see that list comprehension is specified by **square brackets** (just like the list itself) inside which you have a **for** loop over some iterable object. In our example, `new_list` will simply consist of all elements from some_iterable object. The code above is completely equivalent to the one below, however, it takes less space and works a little bit faster!

```
# the equivalent code
new_list = []
for x in some_iterable:
    new_list.append(x)
```

You may wonder why there is a need for list comprehensions at all since we have a `list()` function. Obviously, list comprehensions are used not just for copying elements from some iterable into a list, but mainly for modifying them in some way to create a specific new list. In this case, in the first place of the list comprehension, we write some function of our variable. For example, the code below shows how to create a list of squared numbers.

```
# squared numbers
numbers = [1, 2, 3]
square_list = [x * x for x in numbers]  # [1, 4, 9]
```

Also, we can use list comprehensions to convert elements of a list from one data type to another:

```
# from string to float
strings = ["8.9", "6.0", "8.1", "7.5"]
floats = [float(num) for num in strings]  # [8.9, 6.0, 8.1, 7.5]
```

§1. List comprehension with if

Another way to modify the original iterable object is by introducing the **if statement** into the list comprehension. The basic syntax is this:

```
# list comprehension with condition
new_list = [x for x in some_iterable if condition]
```

The conditional statement allows you to filter the elements of the original collection and work only with the elements you need. The if statement works here as a part of a comprehension syntax. The filtering condition is not an obligatory part, but it can be very useful. For instance, here it is used to create a list of odd numbers from another list:

```
# odd numbers
numbers = [4, 8, 15, 16, 23, 42, 108]
odd_list = [x for x in numbers if x % 2 == 1]  # [15, 23]
```

You can also modify the condition by using standard methods. For instance, if you want to create a list of words that end in *-tion*, you can do it like this:

```
# conditions with functions
text = ["function", "is", "a", "synonym", "of", "occupation"]
words_tion = [word for word in text if word.endswith("tion")]
print(words_tion)  # ["function", "occupation"]
```

Current topic:

- ✓ List comprehension 5★

Topic depends on:

- ✓ Else statement 15★ (Stage 1)
- ✓ For loop 12★ (Stage 1)

Topic is required for:

- ✓ Nested lists
- ✓ Algorithms in Python
- Any and all
- Map and filter
- ✓ Custom generators

Table of contents:

Finally, we can introduce the `else` statement in list comprehension. The syntax here differs a bit: `[x if condition else y for x in some_iterable]`. Using this, we can, for example, get 0 in a new list for each negative number in the old list:

```
old_list = [8, 13, -7, 4, -9, 2, 10]
new_list = [num if num >= 0 else 0 for num in old_list]
print(new_list)  # [8, 13, 0, 4, 0, 2, 10]
```

§2. Recap

In general, list comprehension should be used with caution: where it gains in efficiency it loses in readability. Nevertheless, it is a very useful tool and we hope you'll use it in your programs!

Report a typo

607 users liked this theory. **10** didn't like it. **What about you?**

😍 🙂 😐 ☹️ 😡

Start practicing

Comments (7) Hints (0) Useful links (3) Show discussion