List comprehension in Python – review

Karolina Kotłowska

What are list comprehensions?

According to a definition, list comprehensions is a syntactic construct for creating a list based on existing lists. Python is one of the programming languages that provides this concise method for handling lists. List comprehensions are used for creating new lists from other iterables. The technique is used to effectively create lists to replace 'for' loops and other built-in functions.

Why would we use this technique?

It's highly recommended to use list comprehensions instead of regular loops because you replace many lines of code which makes code is look clearer. What's worth mentioning is that a programmer can use a single line of code to construct a powerful functionality instead of typing all the commands separately (it definitely can take some time).

An example of the "loop way":

```
list=[]
for i in range(10):
    list.append(i/2)
```

An example of the "list comprehension way":

```
list=[i/2 for I in range(10)]
```

What are top 3 advantages of using list comprehension?

One of the biggest advantages is that programmers are allowed to write less code which is often quite better to understand. Also if they are going to pass the code to other developers, the code will be more readable for them.

What's more, there is no use of 'append' attribute while creating a regular loop. That means that our program is much more faster because we do not need to call another function while iterating. It is essential if you work with a lot of lists.

Compared to map, list comprehension is more concise and easier to read. Using map may be a little bit faster, but using list comprehensions will be more obvious what you're doing.

An example:

```
for x, y in list:
    squared = [x ** 2 for x in list]
```

Using map:

```
for x, y in list:
    squared = map(lambda x: x ** 2, numbers)
```

What are the disadvantages of using list comprehension?

Just because it's shorter, it doesn't mean it's always better. Sometimes if your program requires plenty of 'if' and 'break' conditions – we rather use a regular loop to improve clarity of your code.

What is walrus operator and how to use it?

Walrus operator was introduced in Python 3.8. It combines offers an easy way to accomplish two tasks at once. The walrus operator allows us to assign a variable and return the value at once. It also allows us to avoid repetitive code such as: repeated function calls, repeated statements and repeated calls that exhaust iterators.

An example:

```
list=[2,3,4,5,6]
list_length=len(numbers)
list_sum=sum(numbers)
list_description={“length” : list_length, ”sum” : list_sum}
```

Using walrus operator:

```
list=[2,3,4,5,6]
list_description={“length” : (list_length:=len(list)), ”sum” : (list_sum:=sum(list)) }
```

Walrus operator in list comprehensions?

It is quite common to use walrus operator in list comprehensions.

An example:

```
tab=[math.cos(x) for x in s]
result = [x for x in temp if x >= 0]
```

Using walrus operator (avoiding calling cos(x) twice):

```
result = [y for x in s if (y := math.cos(x)) >= 0]
```

When should we use walrus operator?

Walrus operator is not used for optimizing neither the speed of the program nor the algorithm complexity. Hence that, we should use it only if our code will look clearer. If doesn’t – don’t use it.

Bibliography:

https://www.w3schools.com/python/python_lists_comprehension.asp

https://www.programiz.com/python-programming/list-comprehension

https://realpython.com/list-comprehension-python/

https://realpython.com/python-walrus-operator/