Python List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of names, you want a new list, containing only the names with the letter "c" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside like this :

Example

```
names = ["andi", "budi", "ceci", "dedi", "eric"]
namesNewList = []

for name in names:
  if "c" in name:
    namesNewList.append(name)

print(namesNewList) # ['ceci', 'eric']
```

With list comprehension you can do all that with only one line of code:

```
names = ["andi", "budi", "ceci", "dedi", "eric"]

namesNewList = [name for name in names if "c" in name]

print(namesNewlist) # ['ceci', 'eric']
```

Syntax

newlist = [*expression* for *item* in *iterable* if *condition* == True]

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that valuate to True.

Example

Only accept items that are not "dedi":

```
newlist = [name for name in names if name != "dedi"]
```

The condition if name != "dedi" will return True for all elements other than "dedi", making the new list contain all names except "dedi".

The condition is optional and can be omitted:

Example

With no if statement:

```
newlist = [name for name in names]
```

Iterable

The *iterable* can be any iterable object, like a list, tuple, set etc.

Example

You can use the range() function to create an iterable:

```
newlist = [num for num in range(5)]
```

Same example, but with a condition:

Example

Accept only odd numbers :

```
newlist = [num for num in range(5) if num % 2 != 0]
```

Expression

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example

Set the values in the new list to upper case:

```
newlist = [name.upper() for name in names]
```

You can set the outcome to whatever you like:

Example

Set all values in the new list to 'feri':

```
newlist = ['feri' for name in names]
```

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example

Return "feri" instead of "dedi":

```
newlist = [name if name != "dedi" else "feri" for name in names]
```

The *expression* in the example above says:

"*Return the item if it is not dedi, if it is dedi return feri*".

Key Points to Remember

- List comprehension is an elegant way to define and create lists based on existing lists.
- List comprehension is generally more compact and faster than normal functions and loops for creating list.
- However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.
- Remember, every list comprehension can be rewritten in for loop, but every for loop can’t be rewritten in the form of list comprehension.