Department of Computer Science and Engineering (UG Studies)
PES University, Bangalore, India

Introduction to Computing using Python (UE19CS101)

Mr. Prakash C O
Asst. Professor,
Dept. of CSE, PESU
coprakasha@pes.edu

Comprehensions in Python

Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) **using sequences which have been already defined**.

Python supports the following 4 types of comprehensions:

- List Comprehensions
- Dictionary Comprehensions
- Set Comprehensions
- Generator Comprehensions

List comprehension in Python

In set theory, we learn a couple of ways of creating a set.

1. **Enumeration:** All the elements are listed.
   Example: `s = { 1, 3, 5, 7, 9}`

2. **Rule based or builder method:**
   `s = { x | x is odd and 1 <= x <= 10 }`

So far we have learnt how to create lists by enumeration. **In this section, we shall learn how to create lists using rules**. **This method of list creation is called list comprehension**.

We shall see examples from the file **1_list_comprehension.py**

```
l1 = ['hello' for x in range(5)]   # ['hello', 'hello', 'hello', 'hello', 'hello']
```

The above statement is equivalent to the following piece of code.

```
l1 = []
for x in range(5):
      l1.append('hello')
```

List Comprehensions provide an elegant way to create new lists. The following is the basic structure of a list comprehension:

Syntax:

```
output_list = [<output_expr> for <variable> in <iterable> <optional_predicate>]
```

Semantically, this is equivalent the following.

Create an empty list. Execute **for** part of the list comprehension and evaluate the **output_expr** each time. Append the expression result to the list. The result is the new list so created.

Observe the similarity with map.

Example 1: create an output list which contains only the even numbers which are present in the input list.

Using List Comprehension:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
list_using_comp = [x for x in input_list if x % 2 == 0]
print(list_using_comp)
```

Output:

[2, 4, 4, 6]

Using filter() function:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
list_using_filter = filter(lambda x : x % 2 == 0, input_list)
print(list(list_using_filter))
```

Output:

[2, 4, 4, 6]

Using map() function:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
list_using_map = map(lambda x : x % 2 == 0, input_list)
print(list(list_using_map))
```

Output:

[False, True, False, True, True, False, True, False, False]

Let us look at some more examples.

Example 2:
```
squares of numbers from 1 to 5
l2 = [ x * x for x in range(1,6)]
print(l2)
```

Output:

```
[1, 4, 9, 16, 25]
```

Is the above code self explanatory?

Example 3: list of tuples having a number and its square

```
l3 = [ (x, x * x) for x in range(5)]
print(l3)
```

Example 4: list of strings and its length

```
l4 = [ (x, len(x)) for x in ['bangalore', 'mysore', 'hubballi', 'shivamogga']]
print(l4)
```

The one below is an example of nested loops in list comprehension.

Example 5: cartesian product

```
l5 = [ (x, y) for x in range(4) for y in range(4)]
print(l5)
```

Output:

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

The one below is an example of selection amongst the many produced by the loops. Observe the similarity with filter. In fact it is a combination of map and filter.

Example 6: relation: partial order

```
l6 = [ (x, y) for x in range(4) for y in range(4) if x < y]
print(l6)
```

Output:

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

Example 7: convert all words to uppercase

```
# map
a = ['bangalore', 'mysore', 'hubballi', 'shivamogga' ]
b = [ x.upper() for x in a ]
print(b)
```

Output:

```
['BANGALORE', 'MYSORE', 'HUBBALLI', 'SHIVAMOGGA']
```

Example 8: find all words whose len exceeds 7

```
# filter
b = [ x for x in a if len(x) > 7]
print(b)
```

Output:

```
['bangalore', 'hubballi', 'shivamogga']
```

Example 9: convert all words to uppercase if len exceeds 7

```
# combine
b = [ x.upper() for x in a if len(x) > 7]
```

```
print(b)
```
Output:
```
['BANGALORE', 'HUBBALLI', 'SHIVAMOGGA']
```

Example 10: Generate a nested list `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
```
b = [ [x, x+1, x+2] for x in range(1,10,3)]
print(b)
```
Output:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

list comprehension provides an alternate mechanism to functional programming constructs map and filter. We also have set, dict comprehension. These are not part of the course – hence not discussed.

Dict comprehension is defined with a similar syntax, but with a key:value pair in expression.

```
output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}
```

Example 1:

```
state = ['Gujarat', 'Maharashtra', 'Rajasthan']
capital = ['Gandhinagar', 'Mumbai', 'Jaipur']
dict_using_comp = {key:value for (key, value) in zip(state, capital)}
print(dict_using_comp)
```

Output:
```
{'Rajasthan': 'Jaipur', 'Maharashtra': 'Mumbai', 'Gujarat': 'Gandhinagar'}
```

Example 2: create an output dictionary which contains only the odd numbers that are present in the input list as keys and their cubes as values.

```
input_list = [1,2,3,4,5,6,7]
dict_using_comp = {var:var ** 3 for var in input_list if var % 2 != 0}
print(dict_using_comp)
```

Output:
```
{1: 1, 3: 27, 5: 125, 7: 343}
```

References:

1. 18_comprehension_exception.pdf – Prof. N S Kumar, Dept. of CSE, PES University.
2. https://www.w3schools.com/python/
3. https://docs.python.org/
4. https://www.geeksforgeeks.org/comprehensions-in-python/