15. Comprehensions

Comprehensions are a feature of Python which I would really miss if I ever have to leave it. Comprehensions are constructs that allow sequences to be built from other sequences. Three types of comprehensions are supported in both Python 2 and Python 3:

- list comprehensions
- dictionary comprehensions
- set comprehensions
- generator comprehensions

We will discuss them one by one. Once you get the hang of using `list` comprehensions then you can use any of them easily.

15.1. `list` comprehensions

List comprehensions provide a short and concise way to create lists. It consists of square brackets containing an expression followed by a `for` clause, then zero or more `for` or `if` clauses. The expressions can be anything, meaning you can put in all kinds of objects in lists. The result would be a new list made after the evaluation of the expression in context of the `if` and `for` clauses.

Blueprint

```
variable = [out_exp for out_exp in input_list if out_exp == 2]
```

Here is a short example:

```
multiples = [i for i in range(30) if i % 3 == 0]
print(multiples)
# Output: [0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

This can be really useful to make lists quickly. It is even preferred by some instead of the `filter` function. List comprehensions really shine when you want to supply a list to a method or function to make a new list by appending to it in each iteration of the `for` loop. For instance you would usually do something like this:

```
squared = []
for x in range(10):
    squared.append(x**2)
```

You can simplify it using list comprehensions. For example:

```
squared = [x**2 for x in range(10)]
```

15.2. `dict` comprehensions

They are used in a similar way. Here is an example which I found recently:

```
mcase = {'a': 10, 'b': 34, 'A': 7, 'Z': 3}

mcase_frequency = {
    k.lower(): mcase.get(k.lower(), 0) + mcase.get(k.upper(), 0)
    for k in mcase.keys()
}

# mcase_frequency == {'a': 17, 'z': 3, 'b': 34}
```

In the above example we are combining the values of keys which are same but in different typecase. I personally do not use `dict` comprehensions a lot. You can also quickly switch keys and values of a dictionary:

```
{v: k for k, v in some_dict.items()}
```

15.3. `set` comprehensions

They are also similar to list comprehensions. The only difference is that they use braces `{}`. Here is an example:

```
squared = {x**2 for x in [1, 1, 2]}
print(squared)
# Output: {1, 4}
```

15.4. `generator` comprehensions

They are also similar to list comprehensions. The only difference is that they don't allocate memory for the whole list but generate one item at a time, thus more memory effecient.

```
multiples_gen = (i for i in range(30) if i % 3 == 0)
print(multiples_gen)
# Output: <generator object <genexpr> at 0x7fdaa8e407d8>
for x in multiples_gen:
  print(x)
  # Outputs numbers
```