Python Dictionary Comprehension

A comprehension is a compact way of creating a Python data structure from iterators.

With comprehensions, you can combine loops and conditional tests with a less verbose syntax.

Comprehension is considered more Pythonic and often useful in a variety of scenarios.

What is Dictionary Comprehension?

The idea of comprehension is not just unique to lists in Python. Dictionaries also have comprehensions.

Dictionary comprehension is a way to build a new dictionary by applying an expression to each item in an iterable.

It saves you having to write several lines of code, and keeps the readability of your code neat.

Basic Example

Suppose you want to create a dictionary of numbers & squares. You could build such dictionary by inserting one item at a time into an empty dictionary:

```
D = {}
D[0] = 0
D[1] = 1
```

```
D[2] = 4
D[3] = 9
D[4] = 16
print(D)
# Prints {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Or, you could just use an iterator and the range() function:

```
D = {}
for x in range(5):
    D[x] = x**2


print(D)
# Prints {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Here both approaches produce the same result. However, a more Pythonic way to build a dictionary is by using a dictionary comprehension.

The general syntax for a dictionary comprehension is:

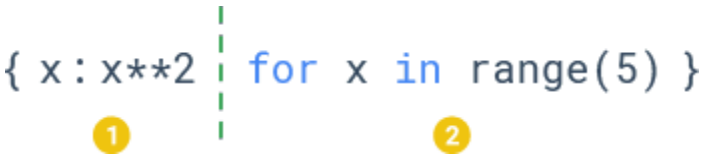
```
{key:value for var in iterable}
```

key: value	*var*	*Iterable*
Key & value can be any expression & evaluated once for each item in iterable	*It takes items from an iterable one by one*	*It's a collection of objects (like a list, tuple etc.)*

Here's how a dictionary comprehension would build the dictionary:

```
D = {x: x**2 for x in range(5)}
print(D)
# Prints {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

In the example above, dictionary comprehension has two parts.



The first part collects the key/value results of expressions on each iteration and uses them to fill out a new dictionary.

The second part is exactly the same as the for loop, where you tell Python which iterable to work on. Every time the loop goes over the iterable, Python will assign each individual element to a variable x.

More Examples

Here are a few examples that might help shed light on the dictionary comprehension.

Example 1

Dictionary comprehensions can iterate over any type of iterable such as lists, strings, files, ranges, and anything else that supports the iteration protocol.

Here's a simple dictionary comprehension that uses string as an iterable.

```
D = {c: c * 3 for c in 'RED'}

print(D)
# Prints {'R': 'RRR', 'E': 'EEE', 'D': 'DDD'}
```

Example 2

In dictionary comprehension, keys can also be computed with expressions just like values. This example calls `lower()` method on keys and `upper()` method on values of a dictionary.

```
L = ['ReD', 'GrEeN', 'BlUe']

D = {c.lower(): c.upper() for c in L}

print(D)
# Prints {'blue': 'BLUE', 'green': 'GREEN', 'red': 'RED'}
```

Extracting a Subset of a Dictionary

Sometimes you want to extract particular keys from a dictionary. This is easily accomplished using a dictionary comprehension.

```
D = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F'}
selectedKeys = [0, 2, 5]

X = {k: D[k] for k in selectedKeys}

print(X)
# Prints {0: 'A', 2: 'C', 5: 'F'}
```

Filter Dictionary Contents

Suppose you want to make a new dictionary with selected keys removed. Here's a simple code that deletes specified keys from a dictionary.

```
D = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F'}
removeKeys = [0, 2, 5]

X = {k: D[k] for k in D.keys() - removeKeys}

print(X)
# Prints {1: 'B', 3: 'D', 4: 'E'}
```

Invert Mapping / Reverse lookup

Given a dictionary `d` and a key `k`, it is easy to find the corresponding value `v = d[k]`. This operation is called a lookup.

But what if you want to retrieve a key `k` using a value `v` in a dictionary? You have to do reverse lookup. This is where dictionary comprehension comes handy.

```
D = {0: 'red', 1: 'green', 2: 'blue'}
```

```
R = {v: k for k,v in D.items()}

print(R)
# Prints {'red': 0, 'green': 1, 'blue': 2}
```

Dictionary Comprehension with Enumerate

Sometimes you want to create a dictionary from the list with list index number as key and list element as value. To achieve this wrap the list in enumerate() function and pass it as an iterable to the dict comprehension.

```
L = ['red', 'green', 'blue']

D = {k:v for k,v in enumerate(L)}

print(D)
# Prints {0: 'red', 1: 'green', 2: 'blue'}
```

Such dictionaries with element index are often useful in a variety of scenarios such as reading a file by lines.

```
D = {ix: line for ix, line in enumerate(open('myFile.txt'))}

print(D)

# {0: 'First line\n',

#  1: 'Second line\n',
#  2: 'Third line\n'}
```

Initialize Dictionary with Comprehension

Dictionary comprehensions are also useful for initializing dictionaries from keys lists, in much the same way as the fromkeys() method. Following example Initializes a dictionary with default value '0' for each key.

```
keys = ['red', 'green', 'blue']


# using dict comprehension

D = {k: 0 for k in keys}

print(D)
```

```
# Prints {'red': 0, 'green': 0, 'blue': 0}


# equivalent to using fromkeys() method
D = dict.fromkeys(keys, 0)
print(D)
# Prints {'red': 0, 'green': 0, 'blue': 0}
```

A standard way to dynamically initialize a dictionary is to combine its keys and values with zip, and pass the result to the dict() function. However, you can achieve the same result with a dictionary comprehension.

```
keys = ['name', 'age', 'job']
values = ['Bob', 25, 'Dev']


# using dict comprehension
D = {k: v for (k, v) in zip(keys, values)}
print(D)
# Prints {'name': 'Bob', 'age': 25, 'job': 'Dev'}


# equivalent to using dict() on zipped keys/values
D = dict(zip(keys, values))
print(D)
# Prints {'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

Dictionary Comprehension with if Clause

A dictionary comprehension may have an optional associated if clause to filter items out of the result.

Iterable's items are skipped for which the if clause is not true.

```
{key:value for var in iterable if_clause}
```

The following example collects squares of even items (i.e. items having no remainder for division by 2) in a range.

```
D = {x: x**2 for x in range(6) if x % 2 == 0}


print(D)
# Prints {0: 0, 2: 4, 4: 16}
```

This dictionary comprehension is the same as a for loop that contains an if statement:

```
D = {}
for x in range(5):
    if x % 2 == 0:
        D[x] = x**2


print(D)
# Prints {0: 0, 2: 4, 4: 16}
```

Nested Dictionary Comprehension

The initial value in a dictionary comprehension can be any expression, including another dictionary comprehension.

```
{key:{dict comprehension} for var in iterable}
```

For example, here's a simple list comprehension that uses a nested for clause.

```
D = {(k,v): k+v for k in range(2) for v in range(2)}
print(D)
# Prints {(0, 1): 1, (1, 0): 1, (0, 0): 0, (1, 1): 2}


# is equivalent to
D = {}
```

```
for k in range(2):
    for v in range(2):
        D[(k,v)] = k+v
print(D)
# Prints {(0, 1): 1, (1, 0): 1, (0, 0): 0, (1, 1): 2}
```