Python List and Dictionary Comprehensions Complete Learning Summary

Udemy 100 Days of Code - Section 26

August 28, 2025

Contents

1 Introduction

This document provides a comprehensive summary of Python list and dictionary comprehensions learned in Section 26 of the 100 Days of Code course. Comprehensions are a Pythonic way to create lists, dictionaries, and other iterables in a concise and readable manner.

2 List Comprehensions

2.1 Basic Concept

List comprehensions provide a concise way to create lists. Instead of using traditional for loops with append operations, we can create lists in a single line of code.

2.1.1 Traditional Approach vs List Comprehension

Traditional approach:

```
numbers = [1, 2, 3]
new_list = []
for n in numbers:
    add_1 = n + 1
    new_list.append(add_1)
```

List comprehension approach:

```
numbers = [1, 2, 3]
new_list = [n + 1 for n in numbers]
```

2.2 Basic Structure

The fundamental structure of a list comprehension is:

`new_list = [new_item for item in list]`

Where:

- `new_item` is the expression to be evaluated for each item
- `item` is the variable representing each element
- `list` is the iterable being processed

2.3 Examples from Your Code

2.3.1 Squaring Numbers

```
numbers = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
squared_numbers = [num ** 2 for num in numbers]
# Output: [1, 1, 4, 9, 25, 64, 169, 441, 1156, 3025]
```

2.3.2 Working with Strings

```
name = 'Anees'
name_list = [letter for letter in name]
# Output: ['A', 'n', 'e', 'e', 's']
```

2.3.3 Using Range

```
double_num = [num * 2 for num in range(1, 5)]
# Output: [2, 4, 6, 8]
```

2.4 Conditional List Comprehensions

You can add conditions to filter elements during the comprehension process.

2.4.1 Structure

```
new_list = [new_item for item in list if test]
```

2.4.2 Examples

Filtering Even Numbers:

```
list_of_strings = ['9', '0', '32', '8', '2', '8', '64', '29', '42', '99
    ']
numbers = [int(num) for num in list_of_strings]
result = [num for num in numbers if num % 2 == 0]
# Output: [0, 32, 8, 2, 8, 64, 42]
```

Filtering by Name Length:

```
names = ['Alex', 'Beth', 'Caroline', 'Dave', 'Eleanor', 'Freddie']
short_names = [name for name in names if len(name) <= 5]
# Output: ['Alex', 'Beth', 'Dave']
```

Combining Transformation and Filtering:

```
cap_names = [name.upper() for name in names if len(name) > 5]
# Output: ['CAROLINE', 'ELEANOR', 'FREDDIE']
```

2.5 Multiple For Loops in List Comprehensions

2.5.1 General Structure

```
[(expression)
 for item1 in iterable1 if condition1
 for item2 in iterable2 if condition2
 ...]
```

2.5.2 Example: Creating Pairs

```
list1 = [1, 2, 3, 4]
list2 = ['a', 'b', 'e', 'i']
result = [(x, y) for x in list1 if x % 2 == 0 for y in list2 if y in '
    aeiou']
# Output: [(2, 'a'), (2, 'e'), (2, 'i'), (4, 'a'), (4, 'e'), (4, 'i')]
```

2.6 Practical Application: File Processing

Your code demonstrates reading from two files and finding common elements:

```
with open('./file1.txt') as file:
    num_list_1 = file.read().splitlines()

with open('./file2.txt') as file:
    num_list_2 = file.read().splitlines()

result = [int(num) for num in num_list_1 if num in num_list_2]
```

3 Dictionary Comprehensions

3.1 Basic Concept

Dictionary comprehensions allow you to create dictionaries in a concise and readable way, similar to list comprehensions but for key-value pairs.

3.2 Basic Structures

3.2.1 From a List

```
new_dict = {new_key:  new_value for item in list}
```

3.2.2 From an Existing Dictionary

```
new_dict = {new_key:  new_value for (key, value) in dict.items()}
```

3.2.3 With Conditions

```
new_dict = {new_key:  new_value for (key, value) in dict.items() if test}
```

3.3 Examples from Your Code

3.3.1 Random Score Generation

```
import random
names = ['Alex', 'Beth', 'Caroline', 'Dave', 'Eleanor', 'Freddie']
score_dict = {f'{name}': random.randint(1, 100) for name in names}
# Example output: {'Alex': 67, 'Beth': 23, 'Caroline': 89, ...}
```

3.3.2 Filtering Passed Students

```
passed_dict = {name: score for (name, score) in score_dict.items() if
    score >= 40}
# Output: Students with scores >= 40
```

3.3.3 Word Length Calculation

```
sentence = "What is the Airspeed Velocity of an Unladen Swallow?"
word_list = sentence.split(" ")
result = {word: len(word) for word in word_list}
# Output: {'What': 4, 'is': 2, 'the': 3, 'Airspeed': 8, ...}
```

3.3.4 Temperature Conversion

```
def convert_C_to_F(temp: int):
    return temp * 9/5 + 32

weather_c = {"Monday": 12, "Tuesday": 14, "Wednesday": 15,
             "Thursday": 14, "Friday": 21, "Saturday": 22, "Sunday":
    24}

weather_f = {day: convert_C_to_F(temp) for (day, temp) in weather_c.
    items()}
# Output: Temperature converted from Celsius to Fahrenheit
```

3.4 Multiple For Loops in Dictionary Comprehensions

3.4.1 General Structure

```
{key_expression: value_expression
 for item1 in iterable1 if condition1
 for item2 in iterable2 if condition2
 ...}
```

3.4.2 Complex Example

```
list1 = [1, 2, 3, 4]
list2 = ['a', 'b', 'e', 'i']
result = {(x, y): f"{x}-{y}" for x in list1 if x % 2 == 0
          for y in list2 if y in 'aeiou'}
# Output: {(2, 'a'): '2-a', (2, 'e'): '2-e', (2, 'i'): '2-i',
#          (4, 'a'): '4-a', (4, 'e'): '4-e', (4, 'i'): '4-i'}
```

4 Nested Comprehensions

4.1 List Comprehension in Dictionary Comprehension

4.1.1 Example: Student Grades Filtering

```python
grades = {
    "Alice": [78, 85, 90, 67],
    "Bob": [88, 72, 95, 80],
    "Charlie": [60, 82, 91, 87]
}

high_grades = {student: [grade for grade in scores if grade > 80]
               for student, scores in grades.items()}
# Output: {'Alice': [85, 90], 'Bob': [88, 95], 'Charlie': [82, 91, 87]}
```

4.2 Dictionary Comprehension in List Comprehension

4.2.1 Example: Word Length Dictionaries

```python
sentences = [
    "hello world",
    "python is fun",
    "list and dict comprehensions"
]

word_length_dicts = [{word: len(word) for word in sentence.split()}
                     for sentence in sentences]
# Output: [{'hello': 5, 'world': 5},
#          {'python': 6, 'is': 2, 'fun': 3},
#          {'list': 4, 'and': 3, 'dict': 4, 'comprehensions': 15}]
```

5 Real-World Application: NATO Alphabet Project

Your NATO alphabet project demonstrates an advanced use of list comprehensions with pandas integration:

```python
import pandas as pd

name = input('Enter your name: ')
nato_data = pd.read_csv('./nato_phonetic_alphabet.csv')

code_list = [row.code for letter in name
             for (_, row) in nato_data.iterrows()
             if letter.upper() == row.letter]
print(code_list)
```

This example shows:

- Multiple for loops in a single comprehension
- Integration with pandas DataFrames
- Conditional filtering based on user input
- Practical application for real-world problems

6 Best Practices and Key Takeaways

6.1 When to Use Comprehensions

- When creating new collections from existing ones
- For simple transformations and filtering operations
- When readability is improved over traditional loops
- For functional programming approaches

6.2 When NOT to Use Comprehensions

- When the logic becomes too complex
- When multiple operations need to be performed
- When debugging is required (traditional loops are easier to debug)
- When side effects are needed (comprehensions should be pure)

6.3 Performance Benefits

Comprehensions are generally faster than equivalent for loops because:

- They are optimized at the C level in CPython
- They avoid repeated method lookups
- They allocate memory more efficiently

6.4 Readability Guidelines

- Keep comprehensions simple and readable
- Use meaningful variable names
- Break complex comprehensions into multiple steps
- Consider using traditional loops for complex logic

7 Summary of Learning Outcomes

Through this section, you have learned:

1. **Basic List Comprehensions**: Creating lists with transformations
2. **Conditional List Comprehensions**: Filtering elements during creation
3. **Multiple For Loops**: Complex iteration patterns
4. **Dictionary Comprehensions**: Creating dictionaries efficiently

5. **Nested Comprehensions**: Combining different comprehension types
6. **Real-world Applications**: Practical projects like NATO alphabet converter
7. **File Processing**: Reading and processing data from files
8. **Data Transformation**: Converting between different data formats
9. **Integration with Libraries**: Using comprehensions with pandas

8 Conclusion

List and dictionary comprehensions are powerful Python features that enable concise, readable, and efficient code. They represent a fundamental shift from imperative to more functional programming paradigms. Your learning progression from basic transformations to complex nested comprehensions with real-world applications demonstrates a solid understanding of these concepts.

The NATO alphabet project particularly showcases the practical application of these concepts in a real-world scenario, combining multiple advanced techniques including pandas integration, file I/O, and complex filtering logic.

Mastering comprehensions is essential for writing Pythonic code and will serve as a foundation for more advanced Python programming concepts.