Building Containers in One Line

Why?

Many of the loops you have written do one thing: build a new list from an old one, keeping some items and changing others. In this activity, you will learn a shorter way to write those loops, and you will decide when the shorter way makes a program clearer.

Prerequisites

- Write `for` loops that build a list with `append()`, a set with `add()`, or a dictionary with `d[key] = value`.
- Use `if` inside a loop to skip some items.
- Use string methods such as `lower()` and `split()`.

Learning Objectives

Content:

- Predict the result of a list, set, or dictionary comprehension, with or without an `if`, by identifying its expression, `for` clause, and condition.
- Rewrite a short accumulator loop as a comprehension, and rewrite a comprehension as a loop.
- Decide whether a given loop can be written as a comprehension, and justify the decision.

Process:

- Match each part of a loop to its place in an equivalent one-line version to infer the general form (Information Processing).
- Sort loops into those that can and cannot become one line, and support each decision with evidence from the code (Critical Thinking).

Model 1: Cleaning the Sign-Up List

A club collected email addresses at an activities fair:

```
signups = ["Ana@jmu.edu", "benji@vt.edu", "Carlos@jmu.edu", "ana@jmu.edu",
           "Dee@uva.edu", "eli@uva.edu", "Fatima@vt.edu", "Gus@jmu.edu"]
```

Each program builds a new list from `signups` two ways: with a loop, and in one line. Both ways give the same result. None of the programs changes `signups`.

Program A

With a loop:

```
lower = []
for email in signups:
    lower.append(email.lower())
```

In one line:

```
lower = [email.lower() for email in signups]
```

Either way, `lower` is:

```
['ana@jmu.edu', 'benji@vt.edu', 'carlos@jmu.edu', 'ana@jmu.edu',
 'dee@uva.edu', 'eli@uva.edu', 'fatima@vt.edu', 'gus@jmu.edu']
```

Program B

With a loop:

```
jmu = []
for email in signups:
    if "@jmu.edu" in email:
        jmu.append(email)
```

In one line:

```
jmu = [email for email in signups if "@jmu.edu" in email]
```

Either way, `jmu` is:

```
['Ana@jmu.edu', 'Carlos@jmu.edu', 'ana@jmu.edu', 'Gus@jmu.edu']
```

Program C

With a loop:

```
names = []
for s in signups:
    if "@jmu.edu" in s:
        names.append(s.split("@")[0])
```

In one line:

```
names = [s.split("@")[0] for s in signups if "@jmu.edu" in s]
```

Either way, `names` is:

```
['Ana', 'Carlos', 'ana', 'Gus']
```

1. In Program A, how is each item in `lower` related to the item at the same position in `signups`?

2. Compare the two versions of Program A. Which parts of the loop version have no matching words in the one-line version, and what in the one-line version does their job instead?

3. Predict the value of `jmu` if the one-line version of Program B began `[email.upper() for ...]` and the rest of the line stayed the same.

4. Both versions of Program C contain the same three pieces: `for s in signups`, `if "@jmu.edu" in s`, and `s.split("@")[0]`. Which piece is in a different place in the one-line version than in the loop, and where did that piece move?

5. The one-line versions are called *list comprehensions*. Using Programs A, B, and C, describe in words what goes in each blank of `[ ___1___ for ___2___ in ___3___ if ___4___ ]`.

6. Rewrite the loop below as a list comprehension.

```
others = []
for email in signups:
    if "@jmu.edu" not in email:
        others.append(email.split("@")[1])
```

7. Predict the value of `[s.split("@")[0].lower() for s in signups if "@jmu.edu" in s]`.

Model 2: Summarizing the Sign-Ups

Each program uses the same `signups` list as Model 1. Programs D and E can be written two ways, like the programs in Model 1. Programs F and G have only a loop version.

Program D

With a loop:

```
domains = set()
for email in signups:
    domains.add(email.split("@")[1])
```

In one line:

```
domains = {email.split("@")[1] for email in signups}
```

Either way, `domains` is (the order may differ):

```
{'jmu.edu', 'uva.edu', 'vt.edu'}
```

Program E

With a loop:

```
contacts = {}
for email in signups:
    contacts[email.split("@")[0].lower()] = email
```

In one line:

```
contacts = {email.split("@")[0].lower(): email for email in signups}
```

Either way, `contacts` is:

```
{'ana': 'ana@jmu.edu', 'benji': 'benji@vt.edu', 'carlos': 'Carlos@jmu.edu',
 'dee': 'Dee@uva.edu', 'eli': 'eli@uva.edu', 'fatima': 'Fatima@vt.edu',
 'gus': 'Gus@jmu.edu'}
```

Program F

```
counts = {}
for email in signups:
    school = email.split("@")[1]
    counts[school] = counts.get(school, 0) + 1
```

Result: `counts` is `{'jmu.edu': 4, 'vt.edu': 2, 'uva.edu': 2}`

Program G

```
for email in signups:
    if "@jmu.edu" not in email:
        print("Visitor:", email)
```

Output:

```
Visitor: benji@vt.edu
Visitor: Dee@uva.edu
Visitor: eli@uva.edu
Visitor: Fatima@vt.edu
```

8. `signups` has eight emails, but `domains` has only three items. Explain why.

9. Compare the one-line version of Program D with the one-line versions in Model 1. What tells Python to build a set instead of a list?

10. In the one-line version of Program E, which part gives the key of each entry, and how can you tell?

11. `signups` has two emails for Ana, but `contacts` has only one entry for `'ana'`. Which of Ana's emails is the value, and why?

12. Program F calls `counts.get()` on the dictionary that the loop is building. Explain why that keeps Program F from being written as a comprehension.

13. How is Program G different from all of the other programs in Models 1 and 2?

14. The one-line versions are called *set comprehensions* and *dictionary comprehensions*. Complete the sentence: A loop can be rewritten as a comprehension when _____.

15. Rewrite the loop below as a dictionary comprehension.

```
lengths = {}
for email in signups:
    if "@vt.edu" in email:
        lengths[email] = len(email)
```

16. Is the one-line version of Program E easier to read than its loop version? Give a reason for your team's answer.

Exercises

Exercises 1 through 3 use this list:

```
words = ["duck", "goose", "duck", "swan"]
```

1. Predict the value of `[w.upper() for w in words if w != "duck"]`.

2. Predict the value of `{w: len(w) for w in words}`.

3. Predict the value of `{len(w): w for w in words}`, and explain why the result has only two entries.

4. Rewrite the loop below as a list comprehension.

```
temps = [50, 23, 68, 32, 95]
celsius = []
for f in temps:
    if f > 32:
        celsius.append(round((f - 32) * 5 / 9))
```

5. Rewrite the loop below as a dictionary comprehension.

```
stock = {"apple": 3, "pear": 0, "plum": 5, "kiwi": 0}
in_stock = {}
for fruit, count in stock.items():
    if count > 0:
        in_stock[fruit] = count
```

6. Rewrite the set comprehension below as a loop, using `signups` from Model 1.

```
initials = {email[0].upper() for email in signups if "@jmu.edu" in email}
```

7. Can the loop below be rewritten as a single comprehension? Explain why or why not, and show how you would shorten the code.

```
long_words = []
count = 0
for w in words:
    if len(w) > 4:
        long_words.append(w)
        count += 1
```

8. A teammate wants to count how many times each word appears in `words`, and writes the code below. The teammate expected `{'duck': 2, 'goose': 1, 'swan': 1}` but got `{'duck': 1, 'goose': 1, 'swan': 1}`. Explain why.

```
counts = {}
counts = {w: counts.get(w, 0) + 1 for w in words}
```

Problem

Write a program that builds a dictionary named `groups` from `signups` in Model 1. Each key should be a domain, and each value should be a list of the names that signed up from that domain. For example, the key `'vt.edu'` should map to the value `['benji', 'Fatima']`. Then explain whether your program is a good candidate for a comprehension.