List Comprehensions

Learning Objectives

- Apply common `for` loop idioms using a list comprehension, including:
  - aggregating,
  - mapping,
  - & filtering

For Loops & List Comprehensions

List Comprehensions

List comprehensions are expressions that generate a list based on the elements of another sequence.

- Succinct way of defining an iteration that builds a list
- Makes it easy to:
  - Copy elements from another sequence
  - Filter elements based on a condition
  - Map elements to new values

List Comprehension Syntax

Recall a `for` loop that copies all characters of a string into a list:

```
new_list = []
for character in "ABCD":
    new_list.append(character)
```

"For each character in the string, place that character in the new list I am creating."

👇 👇 👇

```
new_list = [character for character in "ABCD"]
```

List Comprehension Syntax

A basic list comprehension can be written like so:

```
[<expression> for variable in sequence]
```

- `for variable in sequence` works exactly like a regular `for` loop
  - Each element in `sequence` gets visited one-by-one and is given the name `variable`
- The value of `<expression>` is appended to the output list for each element in the sequence
  - Usually write `<expression>` in terms of `variable`
- A new list is created!

Comprehension vs. Loop

With a comprehension:

```
new_list = [<expression> for variable in sequence]
```

With a loop:

```
new_list = []
for variable in sequence:
    new_list.append(<expression>)
```

Copying Using Comprehensions

Example: create a list containing all of the characters in a string.

```
emoji_string = "🎃🌭🎉"
emoji_list = [emoji for emoji in emoji_string]
```

```
emoji_string = "🎃🌭🎉"
emoji_list = []
for emoji in emoji_string:
    emoji_list.append(emoji)
```

Both snippets produce the same output:

```
["🎃", "🌭", "🎉"]
```

Comprehensions: Filtering

Filter Values Out of a Sequence

We have a basic `for` loop pattern for copying all elements of a sequence that meet a condition. This is called **filtering.**

```
new_list = []                          # [] is a list with no contents
for variable in sequence:              # For each value in the source sequence,
  if condition(variable):              # if that value meets some condition
    new_list.append(<expression>)      # add that value to the end of the new list.
```

`condition()` is a placeholder here to represent some boolean expression that helps decide whether or not to include `value`.

Filter Values Out of a Sequence

```
new_list = []                         # [] is a list with no contents
for variable in sequence:             # For each value in the source sequence,
  if condition(variable):             # if that value meets some condition
    new_list.append(<expression>)     # add that value to the end of the new list.
```

We can rewrite the loop (above) into the comprehension (below)

```
new_list = [<expression> for variable in sequence if condition(variable)]
```

- `<expression> for variable in sequence` works exactly the same way
- `if condition(variable)` allows us to write the expression that is a condition for whether that element of the sequence can be included.

Recall: Getting Non-Zero Exam Scores

```
exam_scores = [100, 0, 89, 93, 78, 67, 0]
non_zeroes = []                # [] is a list with no contents
for score in exam_scores:      # For each score from the list,
  if score > 0:                # if that score is not zero,
    non_zeroes.append(score)   # add that score to the end of the new list.
print(non_zeroes)
```

🖨️👇

```
[100, 89, 93, 78, 67]
```

Recall: Getting Non-Zero Exam Scores

This loop-based version...

```
exam_scores = [100, 0, 89, 93, 78, 67, 0]
non_zeroes = []                # [] is a list with no contents
for score in exam_scores:      # For each score from the list,
  if score > 0:                # if that score is not zero,
    non_zeroes.append(score)   # add that score to the end of the new list.
```

...can be rewritten to:

```
exam_scores = [100, 0, 89, 93, 78, 67, 0]
non_zeroes = [score for score in exam_scores if score > 0]
print(non_zeroes)
```

🖨️ 👇

```
[100, 89, 93, 78, 67]
```

Recall: Checking Capitalization

```
names = ["haRry", "Adi", "molly", "jared", "cEDRIc", "Sukya", "TraviS"]
proper_caps = []                  # [] is a list with no contents
for name in names:                # For each name from the list,
  if name.istitle():              # if that name is in "title case"
    proper_caps.append(name)      # add that name to the end of the new list.
print(proper_caps)
```

🖨️ 👇

```
["Adi", "Sukya"]
```

Recall: Checking Capitalization

This loop-based version...

```
names = ["haRry", "Adi", "molly", "jared", "cEDRIc", "Sukya", "TraviS"]
proper_caps = []                 # [] is a list with no contents
for name in names:               # For each name from the list,
  if name.istitle():             # if that name is in "title case"
    proper_caps.append(name)     # add that name to the end of the new list.
```

...can be rewritten to:

```
names = ["haRry", "Adi", "molly", "jared", "cEDRIc", "Sukya", "TraviS"]
proper_caps = [name for name in names if name.istitle()]
print(names)
```

🖨️ 👇

```
["Adi", "Sukya"]
```

Comprehensions: Mapping

Using the Expression

```
[<expression> for variable in sequence if condition(variable)]
```

- So far, for copying and filtering, we've just had `<expression>` be the `variable` itself
- The `<expression>` can be any expression, though!

A Constant Expression

The expression could be a literal:

```
l = [0 for i in range(10)]
print(l)
```

🖨️👇

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Equivalent to:

```
l = []
for i in range(10):
  l.append(0)
```

More Flexible Expressions

The expression could also be a more complicated set of operations defined *in terms of the* `variable` that we use in the comprehension:

```
exam_scores = [92, 99, 100, 98.5]
curved_scores = [score + 10 for score in exam_scores]
```

This is exactly equivalent to:

```
curved_scores = []
exam_scores = [92, 99, 100, 98.5]
for score in exam_scores:
  curved_scores.append(score + 10)
```

Mapping & Filtering

We can do the mapping and filtering together. Only elements that pass the filter get selected & mapped.

```python
# Get all strings of length 3 and capitalize them.
names = ["hss", "tQm", "aditya", "Sukya"]
capital_initials = [name.upper() for name in names if len(name) == 3]
print(capital_initials)
```

🖨️ ➡️ `["HSS", "TQM"]`

This is equivalent to:

```python
names = ["hss", "tQm", "aditya", "Sukya"]
capital_initials = []
for name in name:
  if len(name) == 3:
    capital_initials.append(name.upper())
```