List comprehensions

Ruth Anderson
UW CSE 160
Autumn 2021
Three Ways to Define a List

• Explicitly write out the whole thing:
  squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

• Write a loop to create it:
  squares = []
  for i in range(11):
      squares.append(i * i)

• Write a list comprehension:
  squares = [i * i for i in range(11)]

• A list comprehension is a concise description of a list
• A list comprehension is shorthand for a loop
List Comprehensions

Simplest Form:

result = [expression for item in sequence]

Examples:

squares = [i * i for i in range(11)]
tens = [x * 10 for x in range(1, 11)]
hundreds = [i * 10 for i in tens]
letters = [x for x in "snow"]
Convert Centigrade to Fahrenheit

ctemps = [17.1, 22.3, 18.4, 19.1]

With a loop:

ftemps = []
for c in ctemps:
    f = celsius_to_farenheit(c)
    ftemps.append(f)

With a list comprehension:

ftemps = [celsius_to_farenheit(c) for c in ctemps]

The comprehension is usually shorter, more readable, and more efficient
Cubes of the first 10 natural numbers

Goal:
Produce: [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]

With a loop:

cubes = []
for x in range(10):
    cubes.append(x ** 3)

With a list comprehension:

cubes = [x ** 3 for x in range(10)]
Powers of 2: ( $2^0$ through $2^{10}$ )

Goal:  $[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]$
Lengths of elements of a list

**Goal:** Write a list comprehension that computes the length of each string in the list `colors`.

```python
colors = ["red", "blue", "purple", "gold", "orange"]
lengths = [**your expression goes here**]
```

⇒ `[3, 4, 6, 4, 6]`
Lengths of elements of a list

**Goal:** Write a list comprehension that computes the length of each string in the list `colors`.

```python
colors = ["red", "blue", "purple", "gold", "orange"]
lengths = [len(x) for x in colors]
```

⇒ `[3, 4, 6, 4, 6]`
List Comprehensions with Conditionals

Can add conditionals:
result = [<expression> for <item> in <sequence> if <condition>]

Example:

squares = [i * i for i in range(11)]
sq_over_ten = [x for x in squares if x > 10]
**Extract values greater than 10**

**Goal:** Given a list, create a list containing ONLY the values from the original list that are greater than 10

**With a loop:**

```python
big_vals = []
for x in input_list:
    if x > 10:
        big_vals.append(x)
```

**With a list comprehension:**

```python
big_vals = [x for x in input_list if x > 10]
```
Even elements of a list

**Goal:** Given an input list `nums`, produce a list of the even numbers in `nums`.

```plaintext
ums = [3, 1, 4, 1, 5, 9, 2, 6, 5]
evens = [**your expression goes here**]

⇒ [4, 2, 6]
```
Even elements of a list

**Goal:** Given an input list `nums`, produce a list of the even numbers in `nums`

```
nums = [3, 1, 4, 1, 5, 9, 2, 6, 5]
⇒ [4, 2, 6]
```

evens = [x for x in nums if x % 2 == 0]
Syntax of a comprehension

\[\{(x, y) \ for \ x \ in \ seq1 \ for \ y \ in \ seq2 \ if \ sim(x, y) > \text{threshold}\}\]

- **expression**
- **for clause (required)** assigns value to the variable \(x\)
- zero or more additional **for** clauses
- zero or more **if** clauses

something that can be iterated
Semantics of a comprehension

result = 
[(x, y) for x in seq1 for y in seq2 if sim(x, y) > threshold]

result = []
for x in seq1:
    for y in seq2:
        if sim(x, y) > threshold:
            result.append((x, y))

... use result ...
Types of comprehensions

List

\[ \{ i \times 2 \text{ for } i \text{ in } \text{range}(3) \} \]

Set

\{ i \times 2 \text{ for } i \text{ in } \text{range}(3) \}

Dictionary

\{ \text{key: value for item in sequence ...} \}
\{ i: i \times 2 \text{ for } i \text{ in } \text{range}(3) \}
**Dictionary of squares**

**Goal:** Given an input list `nums`, produce a dictionary that maps each number to the square of that number.

```python
ums = [3, 1, 4, 5, 9, 2, 6, 7]
square_dict = {**your expression goes here**}
```
Dictionary of squares

**Goal:** Given an input list `nums`, produce a dictionary that maps each number to the square of that number.

```python
nums = [3, 1, 4, 5, 9, 2, 6, 7]
square_dict = {**your expression goes here**}
```

```python
square_dict = {x: x ** 2 for x in nums}
```
Normalize a list

def normalize_list(num_list):
    total = sum(num_list)
    for i in range(len(num_list)):
        num_list[i] = num_list[i] / total
    return num_list
Goal: A list of all possible dice rolls.

With a loop:
rolls = []
for r1 in range(1, 7):
    for r2 in range(1, 7):
        rolls.append((r1, r2))

With a list comprehension:
rolls = [(r1, r2) for r1 in range(1, 7) for r2 in range(1, 7)]
All above-average 2-die rolls

**Goal:** Result list should be a list of 2-tuples:

```
[(2, 6), (3, 5), (3, 6), (4, 4), (4, 5), (4, 6), (5, 3), (5, 4), (5, 5), (5, 6),
 (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)]
```

```
[(r1, r2) for r1 in range(1, 7)
  for r2 in range(1, 7)
  if r1 + r2 > 7]
```

**OR**

```
[(r1, r2) for r1 in range(1, 7)
  for r2 in range(8 - r1, 7)]
```
Sum of above-average 2-die rolls

Goal: Result list should be a list of integers:

\[ [r1 + r2 \text{ for } r1 \text{ in } \text{range}(1, 7) \text{ for } r2 \text{ in } \text{range}(1, 7) \text{ if } r1 + r2 > 7] \]

\[ \Rightarrow [8, 8, 9, 8, 9, 10, 8, 9, 10, 11, 8, 9, 10, 11, 12] \]

Remove Duplicates: Use Set Comprehensions

\[ \{r1 + r2 \text{ for } r1 \text{ in } \text{range}(1, 7) \text{ for } r2 \text{ in } \text{range}(1, 7) \text{ if } r1 + r2 > 7\} \]

\[ \Rightarrow \{8, 9, 10, 11, 12\} \]
Making a Grid

**Goal:** A grid were each element is the sum of it's row # and column #.

(e.g. [[0, 1, 2], [1, 2, 3]])

**With a loop:**

```python
grid = []
for i in range(2):
    row = []
    for j in range(3):
        row.append(i + j)
    grid.append(row)
```

**With a list comprehension:**

```python
grid = [[i + j for j in range(3)] for i in range(2)]
```
A word of caution

List comprehensions are great, but they can get confusing. Err on the side of readability.

```python
nums = [n for n in range(100) if sum([int(j) for j in str(n)]) % 7 == 0]
```
or

```python
nums = []
for n in range(100):
    digit_sum = sum([int(j) for j in str(n)])
    if digit_sum % 7 == 0:
        nums.append(n)
```
A word of caution

List comprehensions are great, but they can get confusing. Err on the side of readability.

```python
nums = [n for n in range(100) if sum([int(j) for j in str(n)]) % 7 == 0]
```

or

```python
def sum_digits(n):
    digit_list = [int(i) for i in str(n)]
    return sum(digit_list)

nums = [n for n in range(100) if sum_digits(n) % 7 == 0]
```