Comprehensions

CS106AP Lecture 24
Today’s questions

How can we write better loops?
What tools do we have for developing code and analyzing data?
Today’s topics

1. Review
2. List and Dict Comprehensions
3. Jupyter Notebooks
4. What’s next?
Review
Tuples
What is a tuple?

('a', 'b', 'c')

('karel', 1)

('simba', 'lion', 25)

like a list, but written with parentheses () instead of []

**Definition**

Tuple

An **immutable** data type for storing values in an ordered linear collection.
Tuples vs. Lists

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- immutable
  - can’t add elements
  - can’t remove elements

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- mutable
  - append(), extend()
  - pop(), remove()
Tuples don’t support item assignment

```python
>>> tup = ('apple', 0.79, 'WA')
```

```python
>>> tup[0]
'apple'
```

```python
>>> tup[2] = 'CA'
```

```
TypeError
```

You can index into to view the elements

You can’t index into to set one of the elements
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup
('apple', 0.79, 'WA')
>>> food, price, location = tup
>>> price
0.79
```
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
>>> a
1
>>> b
2
```

Note: length of left-hand side must = length of right-hand side!
Tuples for swapping variables

```python
>>> a, b = 1, 2
>>> b, a = a, b
>>> b
1
>>> a
2
```

packs \(a, b\) into tuple and unpacks into variables \(b, a\)
<table>
<thead>
<tr>
<th>Tuples</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed number of elements</td>
<td>unbound number of elements</td>
</tr>
<tr>
<td>know # ahead of time</td>
<td>sometimes large number</td>
</tr>
<tr>
<td>sometimes different types</td>
<td>usually all elements are the same type</td>
</tr>
</tbody>
</table>

*not enforced by Python but good style*
Tuples as return values

def animal_min_feedings(animal_dict):
    name, min_feedings = None, float('inf')
    for animal, num_feedings in animal_dict.items():
        if num_feedings < min_feedings:
            name, min_feedings = animal, num_feedings
    return name, min_feedings
**Tuples as return values**

```python
def animal_min_feedings(animal_dict):
    name, min_feedings = None, float('inf')
    for animal, num_feedings in animal_dict.items():
        if num_feedings < min_feedings:
            name, min_feedings = animal, num_feedings
    return name, min_feedings
```

packs name, min_feedings into a tuple and returns!
Tuples as return values

def print_hungriest_animal(filename):
    animal_dict = get_animal_feedings(filename)
    animal, num = animal_min_feedings(animal_dict)
    print(animal, 'eats', num, 'times a day.')

unpacks name, min_feedings
tuple into two variables
Sorting lists with tuples

```python
>>> fruit = [('mango', 3), ('apple', 6), ('lychee', 1), ('apricot', 10)]
>>> sorted(fruit)
[('apple', 6), ('apricot', 10), ('lychee', 1), ('mango', 3)]
```

sorts by the first element in each tuple
How can we write better loops?
Problem: getting a list of squares

- Say you have a list of numbers and you want a list of those numbers’ squares.
Problem: getting a list of squares

- Say you have a list of numbers and you want a list of those numbers’ squares.

[1, 3, 6, 7] --> [1, 9, 36, 49]
Problem: getting a list of squares

- Say you have a list of numbers and you want a list of those numbers’ squares.

  \[1, 3, 6, 7\] \to \[1, 9, 36, 49\]

- How would you produce the output list?
Problem: getting a list of squares - Attempt #1
Problem: getting a list of squares - Attempt #1

# [1, 3, 6, 7] --> [1, 9, 36, 49]
def get_squared(nums):
    result = []
    for n in nums:
        result.append(n ** 2)
    return result
Problem: getting a list of squares - Attempt #1

# [1, 3, 6, 7] --> [1, 9, 36, 49]
def get_squared(nums):
    result = []
    for n in nums:
        result.append(n ** 2)
    return result

nums = [1, 3, 6, 7]
squared_numbers = get_squared(nums)
Problem: getting a list of squares - Attempt #2
Problem: getting a list of squares - Attempt #2

# [1, 3, 6, 7] --> [1, 9, 36, 49]
Problem: getting a list of squares - Attempt #2

# [1, 3, 6, 7] --> [1, 9, 36, 49]

nums = [1, 3, 6, 7]
Problem: getting a list of squares - Attempt #2

# [1, 3, 6, 7] --> [1, 9, 36, 49]

nums = [1, 3, 6, 7]
squared_nums = [n ** 2 for n in nums]
Problem: getting a list of squares - Attempt #2

# [1, 3, 6, 7] --> [1, 9, 36, 49]

```python
nums = [1, 3, 6, 7]
squared_nums = [n ** 2 for n in nums]
```

dthis is a list comprehension!
List Comprehensions

\[ n^2 \text{ for } n \text{ in } \text{nums} \]

**Definition**

List Comprehension

A way to create a list based on existing lists
List Comprehensions

\[ \text{[n} \times \text{n for n in nums]} \]

**Definition**

**List Comprehension**

A way to create a list based on existing lists
List Comprehensions

\[ [n ** 2 \text{ for } n \text{ in } \text{nums}] \]

**Definition**

**List Comprehension**

A way to create a list based on existing lists
List Comprehensions

\[ [n ** 2 \text{ for } n \text{ in } \text{nums}] \]

**Definition**

List Comprehension
A way to create a list based on existing lists
List Comprehensions

\[ n \times 2 \text{ for } n \text{ in } \text{nums} \]

**Definition**

**List Comprehension**
A way to create a list based on existing lists
List Comprehensions

\[ n ** 2 \text{ for } n \text{ in } \text{nums} \]

- Reuses syntax from other features:

**Definition**

**List Comprehension**
A way to create a list based on existing lists.
List Comprehensions

\[ n \times 2 \text{ for } n \text{ in } nums \]

- Reuses syntax from other features:
  - \[ \text{[]} \text{ to create new list} \]

**Definition**

List Comprehension
A way to create a list based on existing lists
List Comprehensions

\[ n \times 2 \text{ for } n \text{ in } \text{nums} \]

- Reuses syntax from other features:
  - [] to create new list
  - foreach loop over other list

**Definition**

*List Comprehension*

A way to create a list based on existing lists
Problem: getting a list of lowercase strings
Problem: getting a list of lowercase strings

- Say you have a list of strings with random casing and you want them to be lowercase.
Problem: getting a list of lowercase strings

- Say you have a list of strings with random casing and you want them to be lowercase.

  in: ['Sonja', 'nichOLAs', 'KYLiE']
  out: ['sonja', 'nicholas', 'kylie']
Problem: getting a list of lowercase strings

- Say you have a list of strings with random casing and you want them to be lowercase.

  in: ['SONja', 'nicHOLAs', 'KYLiE']
  out: ['sonja', 'nicholas', 'kylie']

- How would you produce the output list?
Problem: getting a list of lowercase strings
Problem: getting a list of lowercase strings

strings = [‘SONja’, ‘nicHOLAs’, ‘KYLiE’]
Problem: getting a list of lowercase strings

```python
strings = ['SONja', 'nicHOLAs', 'KYLiE']
lower_strings = [s.lower() for s in strings]
```
Problem: getting a list of lowercase strings

strings = ['SONja', 'nicHOLAs', 'KYLiE']
lower_strings = [s.lower() for s in strings]
Problem: getting a list of lowercase strings

strings = ['SONja', 'nicHOLAs', 'KYLiE']

lower_strings = [s.lower() for s in strings]

Note: a list comprehension creates a new list and doesn’t modify the old list
Problem: getting a list of formatted names
Problem: getting a list of formatted names

- Say you have a list of strings with random casing and you want them to have the first letter uppercase and other letters lowercase.
Problem: getting a list of formatted names

- Say you have a list of strings with random casing and you want them to have the first letter uppercase and other letters lowercase.

  in: ['SONja', 'nicHOLAs', 'KYLiE']

  out: ['Sonja', 'Nicholas', 'Kylie']
Problem: getting a list of formatted names

- Say you have a list of strings with random casing and you want them to have the first letter uppercase and other letters lowercase.

  in: ['SONja', 'nicHOLAs', 'KYLiE']

  out: ['Sonja', 'Nicholas', 'Kylie']

Think/Pair/Share:
How would you produce the output list?
Problem: getting a list of formatted names
Problem: getting a list of formatted names

strings = ['SOnja', 'nicHOLAs', 'KYLiE']
Problem: getting a list of formatted names

strings = ['SOnja', 'nicHOLAs', 'KYLiE']

name_strings = [s[0].upper() + s[1:].lower() for s in strings]
Problem: getting a list of formatted names

```python
strings = ['SONja', 'nicHOLAs', 'KYLiE']
name_strings = [s[0].upper() + s[1:].lower() for s in strings]
```
Problem: getting a list of formatted names

strings = ['SOnja', 'nicHOLAs', 'KYLiE']

name_strings = [s[0].upper() + s[1:].lower() for s in strings]
Problem: getting a list of formatted names

strings = [‘SONja’, ‘nicHOLAs’, ‘KYLiE’]
name_strings = [s[0].upper() + s[1:].lower() for s in strings]
Combining functions with list comprehensions

strings = ['SONja', 'nicHOLAs', 'KYLiE']

name_strings = [s[0].upper() + s[1:].lower() for s in strings]

expression can be decomposed out into a function!
Combining functions with list comprehensions

def name_case(s):
    return s[0].upper() + s[1:].lower()
def name_case(s):
    return s[0].upper() + s[1:].lower()

strings = ['SONja', 'nicHOLAs', 'KYLiE']
Combining functions with list comprehensions

def name_case(s):
    return s[0].upper() + s[1:].lower()

strings = ['Sonja', 'nicHOLAs', 'KYLiE']
name_strings = [name_case(s) for s in strings]
Problem: getting a list of tuples

- Say you have a list of strings. Return a list of tuples where the first element is the string and the second element is the length of the string.
Problem: getting a list of tuples

- Say you have a list of strings. Return a list of tuples where the first elem is the string and the second elem is the length of the string.

  in: ['I', 'love', 'CS106AP']
  out: [('I', 1), ('love', 4), ('CS106AP', 7)]
Problem: getting a list of tuples

- Say you have a list of strings. Return a list of tuples where the first elem is the string and the second elem is the length of the string.

  in: ['I', 'love', 'CS106AP']

  out: [('I', 1), ('love', 4), ('CS106AP', 7)]

Think/Pair/Share: How would you produce the output list?
Problem: getting a list of tuples
Problem: getting a list of tuples

strings = ['I', 'love', 'CS106AP']
Problem: getting a list of tuples

strings = ['I', 'love', 'CS106AP']
tuples = [(s, len(s)) for s in strings]
Problem: getting a list of tuples

strings = ['I', 'love', 'CS106AP']
tuples = [(s, len(s)) for s in strings]
Problem: getting a list of tuples

strings = ['I', 'love', 'CS106AP']
tuples = [(s, len(s)) for s in strings]
Problem: getting a list of tuples

```
strings = ['I', 'love', 'CS106AP']
tuples = [(s, len(s)) for s in strings]
```
Conditions in list comprehensions

```python
>>> nums = [4, 23, 9, 18, 63, 42]
```
Conditions in list comprehensions

```python
>>> nums = [4, 23, 9, 18, 63, 42]
```

what if I only want even numbers?
Conditions in list comprehensions

```python
>>> nums = [4, 23, 9, 18, 63, 42]
>>> even_nums = [n for n in nums if n % 2 == 0]
```
Conditions in list comprehensions

```python
>>> nums = [4, 23, 9, 18, 63, 42]
>>> even_nums = [n for n in nums if n % 2 == 0]
>>> even_nums
```
Conditions in list comprehensions

```python
>>> nums = [4, 23, 9, 18, 63, 42]
>>> even_nums = [n for n in nums if n % 2 == 0]
>>> even_nums
[4, 18, 42]
```
Conditions in list comprehensions

- You can add a condition for additional “filtering”
Conditions in list comprehensions

- You can add a condition for additional “filtering”

[expression for item in list if condition]
Conditions in list comprehensions

- You can add a condition for additional “filtering”

```
[n for n in nums if n % 2 == 0]
```
Conditions in list comprehensions

- You can add a condition for additional “filtering”

\[
[\text{expression for item in list if condition}]
\]

\[
[n \text{ for } n \text{ in nums if } n \% 2 == 0]
\]
Conditions in list comprehensions

- You can add a condition for additional “filtering”

```
[expression for item in list if condition]
```

```
[n for n in nums if n % 2 == 0]
```
Conditions in list comprehensions

- You can add a condition for additional “filtering”

```
[expression for item in list if condition]
```

```
[n for n in nums if n % 2 == 0]
```
Conditions in list comprehensions

- You can add a condition for additional “filtering”

\[
[\text{expression} \ \text{for} \ \text{item} \ \text{in} \ \text{list} \ \text{if} \ \text{condition}]
\]

\[
[n \ \text{for} \ n \ \text{in} \ \text{nums} \ \text{if} \ n \ % \ 2 == 0]
\]
Conditions in list comprehensions

>>> strings = ['sonja', 'kylie', 'nick']
Conditions in list comprehensions

```python
>>> strings = ['sonja', 'kylie', 'nick']
```

what if I don’t want strings that start with ‘s’?
Conditions in list comprehensions

```python
>>> strings = ['sonja', 'kylie', 'nick']

>>> no_s_strings = [s for s in strings if s[0].lower() != 's']
```
Conditions in list comprehensions

```python
>>> strings = ['sonja', 'kylie', 'nick']
>>> no_s_strings = [s for s in strings if s[0].lower() != 's']
>>> no_s_strings
```
Conditions in list comprehensions

```python
>>> strings = ['sonja', 'kylie', 'nick']
>>> no_s_strings = [s for s in strings if s[0].lower() != 's']
>>> no_s_strings
['kylie', 'nick']
```
Conditions in list comprehensions

```python
>>> song = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round']
```
Conditions in list comprehensions

```python
>>> song = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round']

>>> round_words = ['wheels', 'round', 'circle']
```
Conditions in list comprehensions

```python
>>> song = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round']

>>> round_words = ['wheels', 'round', 'circle']

>>> not_round_song = [word for word in song if word not in round_words]
```
Conditions in list comprehensions

```python
>>> song = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round']

>>> round_words = ['wheels', 'round', 'circle']

>>> not_round_song = [word for word in song if word not in round_words]

>>> not_round_song
```
Conditions in list comprehensions

```python
>>> song = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round']

>>> round_words = ['wheels', 'round', 'circle']

>>> not_round_song = [word for word in song if word not in round_words]

>>> not_round_song
['the', 'on', 'the', 'bus', 'go', 'and']
```
Why list comprehensions?
Why list comprehensions?

- They’re more concise
Why list comprehensions?

- They’re more concise
- They’re faster
Why list comprehensions?

- They’re more concise
- They’re faster
- They’re Pythonic
When to **not** use list comprehensions
When to **not** use list comprehensions

- When you need more than one condition
When to **not** use list comprehensions

- When you need more than one condition
- When the expression is complex
When to **not** use list comprehensions

- When you need more than one condition
- When the expression is complex
  - Break it out into a separate function!
Dictionary Comprehensions
Problem: updating all the values in a dict

- Say you have a dictionary of strings to ints, and you want to double the values in the dictionary.
Problem: updating all the values in a dict

- Say you have a dictionary of strings to ints, and you want to double the values in the dictionary.

in: { 'a': 1, 'b': 2, 'c': 3, 'd': 4 }  
out: { 'a': 2, 'b': 4, 'c': 6, 'd': 8 }
Problem: updating all the values in a dict

- Say you have a dictionary of strings to ints, and you want to double the values in the dictionary.

  in: { 'a': 1, 'b': 2, 'c': 3, 'd': 4 }

  out: { 'a': 2, 'b': 4, 'c': 6, 'd': 8 }

- How would you produce the output dictionary?
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

doubled = {k:2*v for (k, v) in d.items()}
Dict Comprehensions

d = \{'a\': 1, 'b': 2, 'c': 3, 'd': 4\}

doubled = \{k:2*v for (k, v) in d.items()\}
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

doubled = {k:2*v for (k, v) in d.items()}
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

doubled = {k:2*v for (k, v) in d.items()}

new key
new value
item
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

doubled = {k:2*v for (k, v) in d.items()}

new key  new value  item  iterable
Problem: reversing the keys and values in a dict

- Make the keys the values and the values the keys!

  in: {‘a’: 1, ‘b’: 2, ‘c’: 3, ‘d’: 4}

Think/Pair/Share:
How would you produce the output dictionary?
Problem: reversing the keys and values in a dict

- Make the keys the values and the values the keys!

  in: {‘a’: 1, ‘b’: 2, ‘c’: 3, ‘d’: 4}


Think/Pair/Share:
How would you produce the output dictionary?

doubled = {k:2*v for (k, v) in d.items()}
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

flipped = {v:k for (k, v) in d.items()}
Dict Comprehensions

```python
d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

flipped = {v:k for (k, v) in d.items()}
```

new key
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

flipped = {v:k for (k, v) in d.items()}
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

flipped = {v:k for (k, v) in d.items()}
Dict Comprehensions

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

flipped = {v:k for (k, v) in d.items()}
What tools do we have for developing code and analyzing data?
Jupyter Notebooks
Jupyter Notebook

- Interactive "notebook" where you can run parts of code
Jupyter Notebook

- Interactive “notebook” where you can run parts of code
  - Can develop code step-by-step
Jupyter Notebook

● Interactive “notebook” where you can run parts of code
  ○ Can develop code step-by-step
  ○ Great for data analysis
Jupyter Notebook

- Interactive “notebook” where you can run parts of code
  - Can develop code step-by-step
  - Great for data analysis
- Built on top of regular Python code
Jupyter Notebook Setup

$ python3 -m pip install jupyter
Jupyter Notebook: Investigating California Air Quality
What’s next?
Lambdas and Custom Sort

- How can we customize our sorting?