Nested Data Structures

CS106AP Lecture 15
Today’s questions

How can we store more information and add more structure to our data?
Today’s topics

1. Review
2. Built-ins
3. Nested data structures
   - Lists
   - Dictionaries
4. What’s next?
Review
Big Picture: Dictionaries + Uniqueness

- A key will only be associated with one value
  - no duplicate keys!
- A dictionary can have multiple values that are the same.

```
+-----+----+
| keys| values |
+-----+----+
| 'hansa'| 3 |
| 'kandula'| 3 |
| 'lumpy'| 1 |
| 'surus' | 3 |
```
Accessing a Dictionary’s Keys

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}

>>> d.keys()

dict_keys(['Gates', 'MemChu', 'Tresidder'])
```

iterable collection of all the keys.
iterable means it can be used in foreach
Accessing a Dictionary’s Keys

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}
>>> list(d.keys())
['Gates', 'MemChu', 'Tresidder']
```

we are using `list()` to convert `d.keys()` into a list
Accessing a Dictionary’s Values

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}
>>> list(d.values())
[23, 116, 57]
```

we are using `list()` to convert `d.values()` into a list
Looping over a Dictionary’s Keys

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}
>>> for building in d.keys():
...     print(building)
...     print(building)
Gates
MemChu
Tresidder
```
Looping over a Dictionary’s Values

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}
>>> for age in d.values():
...     print(age)
23
116
57
```
we can use foreach on the dictionary’s values!
Looping over a Dictionary’s Keys and Values

>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}

>>> for building, age in d.items():
...     print(building, 'is', age, 'years old.'

Gates is 23 years old.

MemChu is 116 years old.

Tresidder is 57 years old.
Printing with sep=

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}
>>> for building, age in d.items():
...     print(building, age, sep=': ')
	Gates: 23
	MemChu: 116
	Tresidder: 57
```

sep is an optional argument like end!
Printing with `sep=`

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}
>>> for building, age in d.items():
...     print(building, age, sep=': ')

Gates: 23
MemChu: 116
Tresidder: 57
```

the separating string will be printed between the arguments you pass into `print()`
Getting a Sorted List of Keys

```python
>>> d = {'Gates': 23, 'Tresidder': 57, 'MemChu': 116}
>>> sorted(d.keys())

['Gates', 'MemChu', 'Tresidder']
```

sorted() returns a list in alphabetical order!
Retrieving Min/Max Values

```python
>>> d = {'Gates': 23, 'MemChu': 116, 'Tresidder': 57}

>>> min(d.values())
23

returns the smallest

>>> max(d.values())
116

returns the biggest
```
**Built-in Function**
A function built into Python that is always available for use.
Examples of Built-ins

print()
input()
str()
int()
float()
len()
open()
list()
sorted()
max()
min()
Built-ins with Lists
Sorted() in Lists

>>> lst = [10, -2, 34, 46, 5]
Sorted() in Lists

>>> lst = [10, -2, 34, 46, 5]
>>> sorted(lst)
Sorted() in Lists

>>> lst = [10, -2, 34, 46, 5]
>>> sorted(lst)

Creates an increasing sorted list
Sorted() in Lists

```python
>>> lst = [10, -2, 34, 46, 5]

>>> sorted(lst)
[-2, 5, 10, 34, 46]
```

Creates an increasing sorted list
Sorted() in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> sorted(lst)
[-2, 5, 10, 34, 46]
>>> lst
```

Creates an increasing sorted list
Sorted() in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> sorted(lst)
[-2, 5, 10, 34, 46]
>>> lst
[10, -2, 34, 46, 5]
```

Creates an increasing sorted list
Sorted() in Lists

>>> lst = [10, -2, 34, 46, 5]
Sorted() in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> sorted(lst, reverse=True)
```
Sorted() in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> sorted(lst, reverse=True)
[46, 34, 10, 5, -2]
```
Sorted() in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> sorted(lst, reverse=True)
[46, 34, 10, 5, -2]
```

You can pass in an optional parameter, reverse=True.
Max/Min in Lists

>>> lst = [10, -2, 34, 46, 5]
Max/Min in Lists

>>> lst = [10, -2, 34, 46, 5]
>>> max(lst)
Max/Min in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> max(lst)
```

*Returns the maximum element in the list*
Max/Min in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> max(lst)
46
```

*Returns the maximum element in the list*
Max/Min in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> max(lst)
46
>>> min(lst)
-2
```
Max/Min in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> max(lst)
46

Returns the minimum element in the list
```
Max/Min in Lists

```python
>>> lst = [10, -2, 34, 46, 5]
>>> max(lst)
46
>>> min(lst)
-2

Returns the minimum element in the list
Max/Min in Lists

>>> lst = ['a', 'b', 'c', 'd']
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)
```
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)

We can use max/min on strings because characters have unicode representations
```
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)
'd'
```

We can use max/min on strings because characters have unicode representations.
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)
'd'
```

We can use `max/min` on strings because characters have unicode representations like `\u0064`, or 100 in decimal.
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)
'd'
>>> min(lst)
```

We can use max/min on strings because characters have unicode representations.
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)
'd'
>>> min(lst)
'a'
```

We can use `max/min` on strings because characters have unicode representations.
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)
'd'
>>> min(lst)
'a'
```

We can use max/min on strings because characters have unicode representations 
\u0061, or 97 in decimal
Max/Min in Lists

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> max(lst)
'd'
>>> min(lst)
'a'
```

We can use max/min on anything where “<” has meaning.
Extending a List

>>> lst = ['a', 'b', 'c', 'd']
Extending a List

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst.extend(['e', 'f'])
```
Extending a List

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst.extend(['e', 'f'])
>>> lst
```
Extending a List

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst.extend(['e', 'f'])
>>> lst
['a', 'b', 'c', 'd', 'e', 'f']
```
Extending a List

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst.extend(['e', 'f'])
>>> lst
['a', 'b', 'c', 'd', 'e', 'f']
```

`extend()` behaves like `+=`
Extending a List

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst.extend(['e', 'f'])
>>> lst
['a', 'b', 'c', 'd', 'e', 'f']
>>> lst += ['g', 'h']
```

`extend()` behaves like `+=`
Extending a List

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst.extend(['e', 'f'])
>>> lst
['a', 'b', 'c', 'd', 'e', 'f']
>>> lst += ['g', 'h']
>>> lst
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

`extend()` behaves like `+=`
Extending a List

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst.extend(['e', 'f'])
>>> lst
['a', 'b', 'c', 'd', 'e', 'f']
>>> lst += ['g', 'h']
>>> lst
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

`extend()` behaves like `+=`
Note on Efficiency

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst += ['e', 'f']

>>> lst = lst + ['e', 'f']

>>> lst = lst + ['e', 'f']
```
Note on Efficiency

>>> lst = ['a', 'b', 'c', 'd']
>>> lst += ['e', 'f']

>>> lst = lst + ['e', 'f']

This creates a new list every time, so when the list gets long, it’s inefficient.
Note on Efficiency

```python
>>> lst = ['a', 'b', 'c', 'd']
>>> lst += ['e', 'f']

>>> lst = lst + ['e', 'f']
```

This modifies in-place, so it’s fast!

This creates a new list every time, so when the list gets long, it’s inefficient.
How can we store more information by adding more structure to our data?
Recall: Animal – Feedings Dictionary

- animal name → number of feedings
- string → int

```
<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>3</td>
</tr>
<tr>
<td>'kandula'</td>
<td>2</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>1</td>
</tr>
<tr>
<td>'surus'</td>
<td>4</td>
</tr>
</tbody>
</table>
```
Recall: Animal – Feedings Dictionary

- animal name → number of feedings
- string → int

What if we wanted to store the times that the animals were fed?
Attempt #1: Animal – Feeding Times Dictionary

- animal name → feeding times
- string → string

What if we wanted to store the times that the animals were fed?
What if we wanted to store the times that the animals were fed?
Attempt #1: Animal – Feeding Times Dictionary

- animal name → feeding times
- string → string

What if we wanted to store the times that the animals were fed?

Times are not easily accessible!
Try #1: Animal – Feeding Times Dictionary

- animal name → **feeding times**
- string → **string**

What if we wanted to store the **times** that the animals were fed? **X** We'd have to call `s.split(',')` anytime we wanted to access a time!
What if we wanted to store the times that the animals were fed?

But those times look like a data type we know of......
Attempt #2: Animal – Feeding Times Dictionary

- animal name → feeding times
- string → list[string]

What if we wanted to store the times that the animals were fed?
Attempt #2: Animal – Feeding Times Dictionary

- animal name → feeding times
- string → list[string]

What if we wanted to store the times that the animals were fed?
Attempt #2: Animal – Feeding Times Dictionary

- animal name → feeding times
- string → list[string]

What if we wanted to store the times that the animals were fed?

We can easily access the individual times!
Nested Data Structures

- We can nest data structures!
Nested Data Structures

- We can nest data structures!
  - Lists in lists
Nested Data Structures

- We can nest data structures!
  - Lists in lists
    - grid/game board
Nested Data Structures

- We can nest data structures!
  - Lists in lists
    - grid/game board
  - Lists in dicts
Nested Data Structures

- We can nest data structures!
  - Lists in lists
    - grid/game board
  - Lists in dicts
    - animals to feeding times
Nested Data Structures

- We can nest data structures!
  - Lists in lists
    - grid/game board
  - Lists in dicts — (assignment 4)
    - animals to feeding times
  - Dicts in dicts
Nested Data Structures

- We can nest data structures!
  - Lists in lists
    - grid/game board
  - Lists in dicts
    - animals to feeding times
  - Dicts in dicts
    - your phone’s contact book
Nested Data Structures

- We can nest data structures!
  - Lists in lists
    - grid/game board
  - Lists in dicts
    - animals to feeding times
  - Dicts in dicts
    - your phone’s contact book
  - ... and so on!
Attempt #2: Animal – Feeding Times Dictionary

- animal name → number of feedings
- string → list[string]

What if we wanted to store the times that the animals were fed?

How do we use this dictionary?
Using a Dictionary Containing a List - Get

Get the feeding times associated with ‘hansa’!
Using a Dictionary Containing a List - Get

```python
>>> d[\'hansa\']
```

Get the feeding times associated with \'hansa\!'
Using a Dictionary Containing a List - Get

```python
>>> d['hansa']
['12:00', '3:00', '9:00']
```

Get the feeding times associated with 'hansa'!
Using a Dictionary Containing a List - Modify Value

```python
>>> d['hansa']
['12:00', '3:00', '9:00']
```

Add a feeding time (‘4:00’) to ‘lumpy’!
Using a Dictionary Containing a List - Modify Value

```python
>>> d['hansa']
['12:00', '3:00', '9:00']
>>> d['lumpy'].append('4:00')
```

Add a feeding time ('4:00') to 'lumpy'!
Using a Dictionary Containing a List - Modify Value

```python
>>> d['hansa']
['12:00', '3:00', '9:00']
>>> d['lumpy'].append('4:00')
```

```
Add a feeding time ('4:00') to 'lumpy'!
```
Using a Dictionary Containing a List - Get Elem

```python
>>> d['hansa']
['12:00', '3:00', '9:00']
```

```python
>>> d['lumpy'].append('4:00')
```

Get the first feeding time for ‘kandula’

```python
gt(12:00, 3:00, 9:00)
```

```
[12:00', '3:00', '9:00']
```

```
['8:00', '1:00']
```

```
['11:00', '4:00']
```

```
['5:00', '3:00', '9:00', '2:00']
```
Using a Dictionary Containing a List - GetElem

```python
>>> d[‘hansa’]
[‘12:00’, ‘3:00’, ‘9:00’]
>>> d[‘lumpy’].append(‘4:00’)
>>> k_times = d[‘kandula’]
```

Get the first feeding time for ‘kandula’
Using a Dictionary Containing a List - Get Elem

```python
>>> d['hansa']
['12:00', '3:00', '9:00']
>>> d['lumpy'].append('4:00')
>>> k_times = d['kandula']
['8:00', '1:00']
```

Get the first feeding time for ‘kandula’
Using a Dictionary Containing a List - Get Elem

```python
>>> d['hansa']
['12:00', '3:00', '9:00']

>>> d['lumpy'].append('4:00')

>>> k_times = d['kandula']
['8:00', '1:00']

>>> k_times[0]

Get the first feeding time for 'kandula'
```
Using a Dictionary Containing a List - Get Elem

```python
>>> d['hansa']
['12:00', '3:00', '9:00']

>>> d['lumpy'].append('4:00')

>>> k_times = d['kandula']
['8:00', '1:00']

>>> k_times[0]
'8:00'
```

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>['12:00', '3:00', '9:00']</td>
</tr>
<tr>
<td>'kandula'</td>
<td>['8:00', '1:00']</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>['11:00', '4:00']</td>
</tr>
<tr>
<td>'suras'</td>
<td>['5:00', '3:00', '9:00', '2:00']</td>
</tr>
</tbody>
</table>

Get the first feeding time for ‘kandula’
Using a Dictionary Containing a List - Get Elem

```python
>>> d[‘hansa’]
[‘12:00’, ‘3:00’, ‘9:00’]
>>> d[‘lumpy’].append(‘4:00’)
>>> k_times = d[‘kandula’]
[‘8:00’, ‘1:00’]
>>> k_times[0]
‘8:00’
>>> d[‘kandula’][0]
```

More concisely,

Get the first feeding time for ‘kandula’
Using a Dictionary Containing a List - Get Elem

```python
>>> d['hansa']
['12:00', '3:00', '9:00']

>>> d['lumpy'].append('4:00')

>>> k_times = d['kandula']
['8:00', '1:00']

>>> k_times[0]
'8:00'

>>> d['kandula'][0]
'8:00'
```

Get the first feeding time for 'kandula'
Using a Dictionary Containing a List - Get Elem

```python
>>> d['hansa']
['12:00', '3:00', '9:00']
>>> d['lumpy'].append('4:00')
>>> k_times = d['kandula']
['8:00', '1:00']
>>> k_times[0]
'8:00'
>>> d['kandula'][0]
'8:00'
```

Get the first feeding time for 'kandula'
Using a Dictionary Containing a List - Set List

```
Using a Dictionary Containing a List - Set List

Reset 'suras' feeding list to ['7:00']
```
Using a Dictionary Containing a List - Set List

```python
>>> d['suras'] = ['7:00']
```

Reset 'suras' feeding list to ['7:00']
Using a Dictionary Containing a List - Set List

```python
>>> d['suras'] = ['7:00']
```

```
<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>['12:00', '3:00', '9:00']</td>
</tr>
<tr>
<td>'kandula'</td>
<td>['8:00', '1:00']</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>['11:00', '4:00']</td>
</tr>
<tr>
<td>'suras'</td>
<td>['7:00']</td>
</tr>
</tbody>
</table>
```

Reset 'suras' feeding list to ['7:00']
Using a Dictionary Containing a List - Set Element

```python
>>> d['surus'] = ['7:00']
```

Set second element in ‘lumpy’ to ‘2:00’
Using a Dictionary Containing a List - Set Element

```python
>>> d['suras'] = ['7:00']
```

```python
>>> lump_list = d['lumpy']
```

Set second element in ‘lumpy’ to ‘2:00’
Using a Dictionary Containing a List - Set Element

```python
>>> d['suras'] = ['7:00']
>>> lump_list = d['lumpy']
>>> lump_list[1] = '2:00'
```

**Set second element in ‘lumpy’ to ‘2:00’**
Using a Dictionary Containing a List - Set Element

```python
>>> d['suras'] = ['7:00']
>>> lump_list = d['lumpy']
>>> lump_list[1] = '2:00'
```

Set second element in ‘lumpy’ to ‘2:00’
Using a Dictionary Containing a List - Set Element

```python
>>> d['suras'] = ['7:00']
>>> lump_list = d['lumpy']
>>> lump_list[1] = '2:00'
# This is the same thing as:
```

**Set second element in ‘lumpy’ to ‘2:00’**
Using a Dictionary Containing a List - Set Element

```python
>>> d['suras'] = ['7:00']
>>> lump_list = d['lumpy']
>>> lump_list[1] = '2:00'
# This is the same thing as:
>>> d['lumpy'][1] = '2:00'
```

Set second element in ‘lumpy’ to ‘2:00’
Think/Pair/Share:

How can we modify our file-reading function to populate the animal – feeding times dictionary?
General Note on Mutability

- Lists and dicts are both mutable data types
General Note on Mutability

- Lists and dicts are both mutable data types
  - We can append or set, and these will modify the original object
General Note on Mutability

- Lists and dicts are both mutable data types
  - We can append or set, and these will modify the original object
  - If we pass a list or a dict into a function and modify it, our changes will persist.
General Note on Mutability

- Lists and dicts are both mutable data types
  - We can append or set, and these will modify the original object
  - If we pass a list or a dict into a function and modify it, our changes will persist. [DEMO]
General Note on Mutability

- Lists and dicts are both mutable data types
  - We can append or set, and these will modify the original object
  - If we pass a list or a dict into a function and modify it, our changes will persist.
- Only immutable types can be used as dictionary keys
General Note on Mutability

- Lists and dicts are both mutable data types
  - We can append or set, and these will modify the original object
  - If we pass a list or a dict into a function and modify it, our changes will persist.
- Only immutable types can be used as dictionary keys
  - e.g. strings, ints, floats, booleans
General Note on Mutability

- Lists and dicts are both mutable data types
  - We can append or set, and these will modify the original object
  - If we pass a list or a dict into a function and modify it, our changes will persist.

- Only immutable types can be used as dictionary keys
  - e.g. strings, ints, floats, booleans
  - Immutable or mutable types can be dictionary values
General Note on Mutability

- Lists and dicts are both mutable data types
  - We can append or set, and these will modify the original object
  - If we pass a list or a dict into a function and modify it, our changes will persist.

- Only immutable types can be used as dictionary keys
  - e.g. strings, ints, floats, booleans

- Immutable or mutable types can be dictionary values
  - e.g. strings, ints, floats, booleans, lists, dictionaries
Think/Pair/Share:

How could we store an animal’s type, diet, and feeding times in a data structure?
Attempt #1: Animal – Info List Dictionary

- animal name → animal type, diet, feeding times
- string → list
### Attempt #1: Animal – Info List Dictionary

- **animal name** → **animal type, diet, feeding times**
- **string** → **list**

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>['elephant', 'grass', '12:00', '3:00', '9:00']</td>
</tr>
<tr>
<td>'kandula'</td>
<td>['elephant', 'grass', '8:00', '1:00']</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>['tortoise', 'kale', '11:00']</td>
</tr>
<tr>
<td>'surus'</td>
<td>['elephant', 'roots', '5:00', '3:00', '9:00', '2:00']</td>
</tr>
</tbody>
</table>
Attempt #1: Animal – Info List Dictionary

- animal name → animal type, diet, feeding times
- string → list

Not super easy to distinguish between the different pieces of data in the list
Dicts in Dicts!

keys

'hanza'
'kandula'
'lumpy'
'surus'

values

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>hansa</td>
<td>1</td>
</tr>
<tr>
<td>kandula</td>
<td>1</td>
</tr>
<tr>
<td>lumpy</td>
<td>1</td>
</tr>
<tr>
<td>surus</td>
<td>1</td>
</tr>
</tbody>
</table>
Attempt #2: Animal – Info Dict Dictionary

- animal name → animal type, diet, feeding times
- string → dict
- use strings as keys to specify what field the values correspond to
Attempt #2: Animal – Info Dict Dictionary

- animal name → animal type, diet, feeding times
- string → dict
- use strings as keys to specify what field the values correspond to
Attempt #2: Animal – Info Dict Dictionary

- animal name → animal type, diet, feeding times
- string → dict
- use strings as keys to specify what field the values correspond to

You can have values of different types.

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>'type': 'elephant', 'diet': 'grass', 'times': ['12:00', '3:00', '9:00']</td>
</tr>
<tr>
<td></td>
<td>'kandula'</td>
</tr>
<tr>
<td></td>
<td>'type': 'elephant', 'diet': 'grass', 'times': ['8:00', '1:00']</td>
</tr>
<tr>
<td></td>
<td>'lumpy'</td>
</tr>
<tr>
<td></td>
<td>'type': 'tortoise', 'diet': 'kale', 'times': ['11:00']</td>
</tr>
<tr>
<td></td>
<td>'surus'</td>
</tr>
<tr>
<td></td>
<td>'type': 'elephant', 'diet': 'roots', 'times': ['5:00', '3:00', '9:00', '2:00']</td>
</tr>
</tbody>
</table>
Attempt #2: Animal – Info Dict Dictionary

- animal name → animal type, diet, feeding times
- string → dict
- use strings as keys to specify what field the values correspond to

**Common pattern**
Using a Dictionary Containing a Dict - Get

```python
>>> d['hansa']

dict

keys

'hansa'
'kandula'
'lumpy'
'surus'

values

'type'    'elephant'
'diet'    'grass'
'times'   ['12:00', '3:00', '9:00']

'type'    'elephant'
'diet'    'grass'
'times'   ['8:00', '1:00']

'type'    'tortoise'
'diet'    'kale'
'times'   ['11:00']

'type'    'elephant'
'diet'    'roots'
'times'   ['5:00', '3:00', '9:00', '2:00']
```
Using a Dictionary Containing a Dict - Get

```python
>>> d['hansa']
{'type': 'elephant',
 'diet': 'grass',
 'times': ['12:00', '3:00', '9:00']}
```
Using a Dictionary Containing a Dict - Get

```python
>>> d['hansa']
{'type': 'elephant',
 'diet': 'grass',
 'times': ['12:00', '3:00', '9:00']}

>>> d['hansa']['type']
'elephant'

>>> d['hansa']['diet']
'grass'

>>> d['hansa']['times']
['12:00', '3:00', '9:00']

>>> d['hansa']['type']
'elephant'

>>> d['hansa']['diet']
'grass'

>>> d['hansa']['times']
['8:00', '1:00']

>>> d['lumpy']['type']
'tortoise'

>>> d['lumpy']['diet']
'kale'

>>> d['lumpy']['times']
['11:00']

>>> d['surus']['type']
'elephant'

>>> d['surus']['diet']
'roots'

>>> d['surus']['times']
['5:00', '3:00', '9:00', '2:00']
```
Using a Dictionary Containing a Dict - Get

```python
>>> d['hansa']
{'type': 'elephant',
 'diet': 'grass',
 'times': ['12:00', '3:00', '9:00']}

>>> d['hansa']['type']
'elephant'
```

```
{ 'type': 'elephant',
  'diet': 'grass',
  'times': ['12:00', '3:00', '9:00']}
```

```python
>>> d['hansa']['type']
'elephant'
```

```
'elephant'
```

```python
>>> d['hansa']['diet']
'grass'
```

```
'grass'
```

```python
>>> d['hansa']['times']
['12:00', '3:00', '9:00']
```

```
['12:00', '3:00', '9:00']
```

```
'person'
```

```
'person'
```

```
'type': 'elephant',
'diet': 'grass',
'times': ['12:00', '3:00', '9:00']}
```

```
'type': 'elephant',
'diet': 'grass',
'times': ['12:00', '3:00', '9:00']}
```

```
'type': 'elephant',
'diet': 'grass',
'times': ['12:00', '3:00', '9:00']}
```
Using a Dictionary Containing a Dict - Get

```python
>>> d['hansa']
{'type': 'elephant',
 'diet': 'grass',
 'times': ['12:00', '3:00', '9:00']}
>>> d['hansa']['type']
'elephant'
>>> d['hansa']['times']
['12:00', '3:00', '9:00']
```
Using a Dictionary Containing a Dict - Get

```python
>>> d['hansa']
{'type': 'elephant',
 'diet': 'grass',
 'times': ['12:00', '3:00', '9:00']}

>>> d['hansa']['type']
'elephant'

>>> d['hansa']['times']
['12:00', '3:00', '9:00']
```
Using a Dictionary Containing a Dict - Set

```python
# for animal 'sky'
>>> new_dict = {}
```

```python
keys

<table>
<thead>
<tr>
<th>'hansa'</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'elephant'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'grass'</td>
</tr>
<tr>
<td>'times'</td>
<td>['12:00', '3:00', '9:00']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>'kandula'</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'elephant'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'grass'</td>
</tr>
<tr>
<td>'times'</td>
<td>['8:00', '1:00']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>'lumpy'</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'tortoise'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'kale'</td>
</tr>
<tr>
<td>'times'</td>
<td>['11:00']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>'surus'</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'elephant'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'roots'</td>
</tr>
<tr>
<td>'times'</td>
<td>['5:00', '3:00', '9:00', '2:00']</td>
</tr>
</tbody>
</table>
```
Using a Dictionary Containing a Dict - Set

# for animal 'sky'

```python
>>> new_dict = {}
>>> new_dict['type'] = 'chicken'
```
# for animal 'sky'

```python
>>> new_dict = {}
>>> new_dict['type'] = 'chicken'
>>> new_dict['diet'] = 'grass'
```

![Diagram showing keys and values of a dictionary containing a dict for each animal]
Using a Dictionary Containing a Dict - Set

# for animal 'sky'

```python
>>> new_dict = {}  
>>> new_dict['type'] = 'chicken'  
>>> new_dict['diet'] = 'grass'  
>>> new_dict['times'] = ['4:00']  
```

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'elephant'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'grass'</td>
</tr>
<tr>
<td>'times'</td>
<td>['12:00', '3:00', '9:00']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'elephant'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'grass'</td>
</tr>
<tr>
<td>'times'</td>
<td>['8:00', '1:00']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'tortoise'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'kale'</td>
</tr>
<tr>
<td>'times'</td>
<td>['11:00']</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'type'</td>
<td>'elephant'</td>
</tr>
<tr>
<td>'diet'</td>
<td>'roots'</td>
</tr>
<tr>
<td>'times'</td>
<td>['5:00', '3:00', '9:00', '2:00']</td>
</tr>
</tbody>
</table>
# for animal 'sky'

```python
>>> new_dict = {}
>>> new_dict['type'] = 'chicken'
>>> new_dict['diet'] = 'grass'
>>> new_dict['times'] = ['4:00']
>>> new_dict
```

```
{'type': 'chicken', 'diet': 'grass', 'times': ['4:00']}
```

```python
>>> new_dict = {}
>>> new_dict['type'] = 'elephant'
>>> new_dict['diet'] = 'grass'
>>> new_dict['times'] = ['12:00', '3:00', '9:00']
>>> new_dict
```

```
{'type': 'elephant', 'diet': 'grass', 'times': ['12:00', '3:00', '9:00']}
```

```python
>>> new_dict = {}
>>> new_dict['type'] = 'tortoise'
>>> new_dict['diet'] = 'kale'
>>> new_dict['times'] = ['11:00']
>>> new_dict
```

```
{'type': 'tortoise', 'diet': 'kale', 'times': ['11:00']}
```

```python
>>> new_dict = {}
>>> new_dict['type'] = 'elephant'
>>> new_dict['diet'] = 'roots'
>>> new_dict['times'] = ['5:00', '3:00', '9:00', '2:00']
>>> new_dict
```

```
{'type': 'elephant', 'diet': 'roots', 'times': ['5:00', '3:00', '9:00', '2:00']}
```
Using a Dictionary Containing a Dict - Set

```python
# for animal 'sky'
>>> new_dict = {}
>>> new_dict['type'] = 'chicken'
>>> new_dict['diet'] = 'grass'
>>> new_dict['times'] = ['4:00']
>>> new_dict
{'type': 'chicken', 'diet': 'grass', 'times': ['4:00']}
```

```plaintext
keys          values
---           -------
hansa         'type': 'elephant',
               'diet': 'grass',
               'times': ['12:00', '3:00', '9:00']
kandula       'type': 'elephant',
               'diet': 'grass',
               'times': ['8:00', '1:00']
lumpy         'type': 'tortoise',
               'diet': 'kale',
               'times': ['11:00']
surus         'type': 'elephant',
               'diet': 'roots',
               'times': ['5:00', '3:00', '9:00', '2:00']
```
# for animal ‘sky’

```python
>>> new_dict = {}
>>> new_dict['type'] = 'chicken'
>>> new_dict['diet'] = 'grass'
>>> new_dict['times'] = ['4:00']
>>> new_dict
{'type': 'chicken', 'diet': 'grass', 'times': ['4:00']}
>>> d['sky'] = new_dict
```

```
<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>hansa</td>
<td>{'type': 'elephant', 'diet': 'grass', 'times': ['12:00', '3:00', '9:00']}</td>
</tr>
<tr>
<td>kandula</td>
<td>{'type': 'elephant', 'diet': 'grass', 'times': ['8:00', '1:00']}</td>
</tr>
<tr>
<td>lumpy</td>
<td>{'type': 'tortoise', 'diet': 'kale', 'times': ['11:00']}</td>
</tr>
<tr>
<td>surus</td>
<td>{'type': 'elephant', 'diet': 'roots', 'times': ['5:00', '3:00', '9:00', '2:00']}</td>
</tr>
</tbody>
</table>
```
Using a Dictionary Containing a Dict - Set

# for animal 'sky'
>>> new_dict = {}
>>> new_dict['type'] = 'chicken'
>>> new_dict['diet'] = 'grass'
>>> new_dict['times'] = ['4:00']
>>> new_dict
{'type': 'chicken', 'diet': 'grass', 'times': ['4:00']}
>>> d['sky'] = new_dict

```
{
    'type': 'chicken',
    'diet': 'grass',
    'times': ['4:00']
}
```
Nested Data Structures Overview

- We can have lists in lists, dicts in lists, dicts in dicts, and so on...
Nested Data Structures Overview

- We can have lists in lists, dicts in lists, dicts in dicts, and so on...
- Lists and dicts are mutable (and can’t be used as keys)
Nested Data Structures Overview

● We can have lists in lists, dicts in lists, dicts in dicts, and so on...

● Lists and dicts are mutable (and can’t be used as keys)

● Nesting data structures can help us store even more information in a structured manner!
What’s next?
Roadmap

Day 1!

Programming Basics

The Console

Images

Data structures

Midterm

Graphics

Object-Oriented Programming

Everyday Python

Life after CS106AP!