Dictionaries

CS 1111
Introduction to Programming
Spring 2019
How do Computer Programs Fit in with the World Around Them?

- Thing (object type): Hotel
- Thing (object type): Car
- Thing (object type): Car
- Thing (object type): Car
Objects and Properties

Object type: Hotel
- Properties
  - Name: Awesome
  - Rating: 5
  - Rooms: 70
  - Bookings: 56
  - Pool: true
  - Gym: true

Object type: Car
- Properties
  - Make: UVA1
  - currentSpeed: 30mph
  - Color: yellow
  - Fuel: gasoline

Object type: Car
- Properties
  - Make: UVA2
  - currentSpeed: 20mph
  - Color: red
  - Fuel: gasoline

Object type: Car
- Properties
  - Make: UVA2
  - currentSpeed: 35mph
  - Color: blue
  - Fuel: gasoline
Overview: Dictionaries

• Dictionary = unordered sequence of data
  • Python 3.6 remembers order of items in dictionary
• Mutable data type

• Each element in a dictionary consists of 2 parts: Key-value pair

• Key = index to locate a specific value
• Deterministic:
  • A particular key can only have one value

• Example
  • key = currentSpeed, value = 30mph
  • key = student ID, value = student name
Example: Dictionaries

hotel_dict

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>Awesome</td>
</tr>
<tr>
<td>Rating:</td>
<td>5</td>
</tr>
<tr>
<td>Rooms:</td>
<td>70</td>
</tr>
<tr>
<td>Bookings:</td>
<td>56</td>
</tr>
<tr>
<td>Pool:</td>
<td>true</td>
</tr>
<tr>
<td>Gym:</td>
<td>true</td>
</tr>
</tbody>
</table>

car1_dict

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make:</td>
<td>UVA1</td>
</tr>
<tr>
<td>currentSpeed:</td>
<td>30mph</td>
</tr>
<tr>
<td>Color:</td>
<td>yellow</td>
</tr>
<tr>
<td>Fuel:</td>
<td>gasoline</td>
</tr>
</tbody>
</table>

car2_dict

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make:</td>
<td>UVA2</td>
</tr>
<tr>
<td>currentSpeed:</td>
<td>20mph</td>
</tr>
<tr>
<td>Color:</td>
<td>red</td>
</tr>
<tr>
<td>Fuel:</td>
<td>gasoline</td>
</tr>
</tbody>
</table>

car3_dict

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make:</td>
<td>UVA2</td>
</tr>
<tr>
<td>currentSpeed:</td>
<td>35mph</td>
</tr>
<tr>
<td>Color:</td>
<td>blue</td>
</tr>
<tr>
<td>Fuel:</td>
<td>gasoline</td>
</tr>
</tbody>
</table>
Another Example

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Theresa</td>
</tr>
<tr>
<td>address</td>
<td>17 Binnacle Ln Qui</td>
</tr>
<tr>
<td>phone</td>
<td>479-4923</td>
</tr>
</tbody>
</table>

[Images from https://en.wikipedia.org/wiki/Telephone_directory]
# Lists vs. Dictionaries

**Lists**

- Complex type
- Mutable
- **Ordered** sequence of data
- **Index** = 0, 1, 2, ...

<table>
<thead>
<tr>
<th>index</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Dictionaries**

- Complex type
- Mutable
- **Unordered** sequence of data (until Python 3.6),
- **Index** = *user-defined* key
- Unique key

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dictionaries

Create a dictionary

```python
phonebook = {'friend1': '111-1111', 'friend2': '222-2222'}

phonebook2 = {}
```

Empty dictionary

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'friend1'</td>
<td>'111-1111'</td>
</tr>
<tr>
<td>'friend2'</td>
<td>'222-2222'</td>
</tr>
</tbody>
</table>
Exercise: Create Dictionary with {}

- Create a dictionary of a “friend” object.

- You will start by getting inputs from 5 friends (neighbors). Inputs contain
  - Name
  - Email address

- Use {} to create a “friends” dictionary with the information you gathered

- Print the dictionary content using
  
  \[\text{print(}\text{your-dictionary-name}\text{)}\]

Reminder: Dictionary does not allowed duplicate key
Access items from a Dictionary

Retrieve a value from a dictionary

```
phonebook['friend1']
```

Include quotations for string keys

```
Dictionary_name[key]
```

What would happen if we try to access a key that does not exist?
Exercise: Access Items with [key]

- Revisit your “friends” dictionary
- Access 2 friends and print their email addresses
- Try accessing a friend who is not in the dictionary and observe what happens
- Print the dictionary content using
  
  \[
  \text{print(} \text{your-dictionary-name})
  \]
Add Items to a Dictionary

```python
phonebook = {'friend1': '111-1111', 'friend2': '222-2222', 'friend3': '333-3333'}

phonebook['friend4'] = '444-4444'

phonebook['friend1'] = '555-5555'
```

**Dictionary**

\[ \text{Dictionary}_\text{name}[\text{key}] = \text{value} \]

- No duplicate keys in a dictionary
- When you assign a value to an existing key, the new value replaces the existing value
Exercise: Add Items with \texttt{[key]} \\

\begin{itemize}
\item Revisit your “friends” dictionary
\item Add 2 more friends and their email addresses to the dictionary
\item Try adding one more friend with the key already in the dictionary and observe what happens ( \ldots reassign the value)
\item Print the dictionary content using
\begin{verbatim}
print(your-dictionary-name)
\end{verbatim}
\end{itemize}
Delete Items from Dictionaries

```python
del phonebook['friend1']
```

`del` deletes an element at a particular position

```python
phone_number = phonebook.pop('friend1')
```

`pop()` gets a value (and use it somewhere else), and deletes an element (a key/value pair)

What would happen if we try to delete an item with an index that doesn’t exist?
Exercise: Remove Item with `del` and `pop()`

- Revisit your “friends” dictionary

- Remove one friend from the dictionary, using `del`

- Print the dictionary content using `print(your-dictionary-name)`

- Try removing a friend whose name is not in the dictionary, using `del`, and observe what happens

- Remove one friend from the dictionary, using `pop()`

- Print the dictionary content using `print(your-dictionary-name)`

- Try removing a friend whose name is not in the dictionary, using `pop()`, and observe what happens
Length of Dictionaries

```
phonebook = {'friend1': '111-1111',
             'friend2': '222-2222',
             'friend3': '333-3333'}

num_items = len(phonebook)
```

`len()` is a function to return the length of a dictionary (i.e., the number of items in a dictionary)
Exercise: Get Size with `len(dict)`

- Revisit your “friends” dictionary
- Print the number of items of the dictionary
- Print the dictionary content using
  ```python
  print(your-dictionary-name)
  ```
Retrieve Values, Keys, or Items

# retrieve a value for a particular key
phonebook.get("friend4")

# access a non-existent key, set return value
phonebook.get("friend99", "friend99 does not exist")

phonebook.items()    # retrieve all the keys and values
phonebook.keys()     # retrieve all the available keys
phonebook.values()   # retrieve all the values

get(key, optional-msg) gets a particular value based on key
items() gets all the keys and values
keys() gets all the keys
values() gets all the values
Exercise: Retrieve Value with `get()`

- Revisit your “friends” dictionary
- Print the dictionary content using
  ```python
prompt(print(your-dictionary-name))
  ```
- Retrieve an email address of one friend, using `get()`, and print it
- Try retrieving an email of a friend whose name is not in the dictionary, using `get()`, and observe what happens
- Try (again) retrieving an email of a friend whose name is not in the dictionary, using `get()`, set return value if the friend’s name (key) is not found, and observe what happens
Exercise: Retrieve Items, Keys, Values

- Revisit your “friends” dictionary
- Print the dictionary content using
  
  \[
  \text{print(} \text{your-dictionary-name)}\]
- Retrieve all items from the dictionary using \text{items()}, and print them
- Retrieve all keys from the dictionary using \text{keys()}, and print them
- Retrieve all values from the dictionary using \text{values()}, and print them
Mix Data Types in Dictionaries

test_scores = {'friend1': [88, 92, 100],
               'friend2': [95, 88, 81],
               'friend3': [70, 75, 78]}

print(test_scores)
print('friend2\'s scores: ' + str(test_scores[‘friend2’]))
# why do we need str()? 

friend3_scores = test_scores[‘friend3’]
print('friend3\'s scores: ' + str(friend3_scores))

Keys must be unique and immutable (primitive data type)
Values can be of any data types
Exercise: List in Dictionary

- You will now work with a dictionary that has mixed types of content.

- Gather some more information from friends. You will create a list of the information. Such as
  - List of email addresses, or
  - List of phone numbers, or
  - List of favorite cartoons (or movies), or
  - List of courses currently taken, or
  - List of anything you are interested to know about your friends

- Create a “favoritefriends” dictionary, using the friend’s name as key and a list of the information you gather as value for that friend

- Print the dictionary

- Access 2 friends in the “favoritefriends” dictionary and print the corresponding values
in

phonebook = {'friend1': '111-1111',
             'friend2': '222-2222',
             'friend3': '333-3333'}

phonebook['upsorn'] = '444-4444'
print('upsorn' in phonebook)
print('upsorn' not in phonebook)
print('upsorn' in phonebook.keys())
print('444-4444' in phonebook.values())

in is a keyword and can be used to check if a particular item/key/value is in the dictionary/keys/values
Empty the Dictionaries

`phonebook.clear()`

`clear()` empties the dictionary
Suppose we are using a dictionary to keep track of the number of animals in a small pet store.

Variables

```
numAnimals = {}
numAnimals['cat'] = 3
numAnimals['fish'] = 22
numAnimals['dog'] = 5
```

Heap

```
<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'cat'</td>
<td>3</td>
</tr>
<tr>
<td>'fish'</td>
<td>22</td>
</tr>
<tr>
<td>'dog'</td>
<td>5</td>
</tr>
</tbody>
</table>
```
Tracing through Code with Dictionaries

Suppose we are using a dictionary to keep track of the number of animals in a small pet store.

```python
numAnimals = {}
numAnimals['cat'] = 3
numAnimals['fish'] = 22
numAnimals['dog'] = 5

print(numAnimals['dog'])
print(numAnimals[2])
print(numAnimals.keys())
print(numAnimals.values())
```

- `numAnimals` is a dictionary.
- The dictionary contains entries for 'cat', 'fish', and 'dog'.
- The number of each type of animal is as follows: cat: 3, fish: 22, dog: 5.
- The code prints the number of dogs, the second element of the dictionary, the keys, and the values.

**Variables**
- `numAnimals` is an object.
- The keys are 'cat', 'fish', and 'dog'.
- The values are 3, 22, and 5.

**Heap**
- The keys are stored in an array on the heap.
- The values are stored in another array on the heap.
Tracing through Code with Dictionaries

Suppose we are using a dictionary to keep track of the number of animals in a small pet store

```python
numAnimals = {}
numAnimals['cat'] = 3
numAnimals['fish'] = 22
numAnimals['dog'] = 5

print(numAnimals['dog'])
print(numAnimals.keys())
print(numAnimals.values())

numAnimals['bird'] = 4
numAnimals['cat'] = 2
```

Variables

```
numAnimals
```

Heap

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'cat'</td>
<td>3  2</td>
</tr>
<tr>
<td>'fish'</td>
<td>22</td>
</tr>
<tr>
<td>'dog'</td>
<td>5</td>
</tr>
<tr>
<td>'bird'</td>
<td>4</td>
</tr>
</tbody>
</table>

A100 of 'cat'
**Dictionaries (wrap up)**

```python
dict = {}
dict['1'] = 'cat'
dict['dog'] = -8
dict[False] = 'squirrel'
```

- declare a dictionary with curly braces
- add to a dict by specifying a key and assigning it a value
- a key must be immutable (no lists)
- the `.keys()` method returns all the keys (but we can’t rely on an order)
- the `.values()` method returns all the values (but we can’t rely on an order)
- assigning to a key that already has that value overwrites the old value

```python
dict['dog'] = 5
dict['dog'] = 5
```

```python
if 'dog' in dict.keys():
    print('dog has a mapping!')
if 'cat' in dict.keys():
    print('cat has a mapping!')
```

```python
print(dict)
```
Exercise

• Create a dictionary of an “experience” object.

• You will start by getting inputs from users. Inputs contain:
  • The name of the experience (e.g., "software engineer")
  • The company of the experience (e.g., “IBM”)
  • The year of the experience (e.g., “1996”)

• Add the users’ inputs to an “experience” dictionary:
  • The keys in the dictionary will be the year of the experience, while the values will be the name of the experience and the companies, stored as a list.
  • E.g., { ‘1996’ : ['software engineer', 'IBM'], ‘1993’ : ['sale', 'Target'] }

• You should get at least 2 experience inputs from the users.

• Print each experience in a separate line

• You may assume that all experiences passed in as arguments never have two experiences with the same company and year.

• Try to add more actions: retrieve items, delete items, update items, ...
Summary

• Must know (based on exam2 topic list, as of 03/17/2019)
  • mapping.keys()
  • mapping.values()
  • mapping.items()
  • mapping.pop(key)

(mapping refer to a variable of dict type)