Dictionaries in Python

L435/L555

Dept. of Linguistics, Indiana University
Fall 2016
Dictionaries

Dictionaries

Dictionaries are a complex data structure that stores pairs of values. A pair consists of a **key** and a **value**. Each key can be in the dictionary only once.

- Key-value pairs (items) are created by these **mappings**

Syntax:

```python
lex = {}   # empty dictionary
lex = { 'man': 'Mann', 'woman': 'Frau',
        'child': 'Kind', 'dog': 'Toele'}
lex[ 'dog' ] = 'Hund'  # change an element
lex[ 'cat' ] = 'Katze' # add an element
print(lex[ 'woman' ])  # access an element
```

A minimal dictionary

```
english2arabic = {'man': 'rajul',
                 'country': 'balad',
                 'peace': 'salam',
                 'terror': 'irhab',
                 'child': 'tifl'}
```

When printed, the order of elements may come out differently

- Dictionaries are like sets: order is unpredictable!
Dictionaries are indexed by keys, not by digits indicating position as in lists

- No duplicate keys/entries are allowed
- `len()` checks the number of entries

The key has to be an immutable type

- A tuple can be a key, but not a list

The values can be anything.

If the key is not present, you get an error, so you often need a check:

```python
if 'woman' in english2arabic:
    print(english2arabic['woman'])
```

- Note our old friend `in` in this example!
Accessing a key’s value

As an alternative to bracket access, you can use `.get()`

- `english2arabic.get[’woman’]` throws an error
- `english2arabic.get(’woman’)` returns `None`
- `english2arabic.get(’woman’,’UNKNOWN’)` returns ‘UNKNOWN’
Dictionaries support iteration:

```python
dict = { 'NN': 5, 'PRP': 13, 'VBZ': 4}
for key in dict:
    print('{} occurred {} times'.format(key, dict[key]))
```

```python
dict = { 'NN': 5, 'PRP': 13, 'VBZ': 4}
for key, val in dict.items():
    print('{} occurred {} times'.format(key, val))
```

**Iteration**

Caution: The loop will make sure that you will look at every key, but not in which order.
Accessing items

- Dictionaries support the in operator (tests for keys, not values)
- Accessing items:
  - The .values() method gets us all the values.
  - The .keys() method gets us all the keys.
  - The .items() method gets us all the items.
(Reverse) lookup

It is easy to look up a value for a key

- What if we want to look up the (first) key associated with a value?
- What if we want to find all keys associated with a value?
Other dictionary methods

- **pop (see also: del)**
  gets the value for a given key and deletes the pair from the dictionary

  ```python
  val = english2arabic.pop('man')
  ```

- **copy**
  returns a new dictionary with the same values

  ```python
  x = {'Sandra': 852, 'Damir': 850, 'lab': 306}
  y = x.copy()
  ```

- **update**
  updates values of pairs from another dictionary

  ```python
  z = {'Markus': 851}
  x.update(z)
  ```
Building dictionaries

- It is very tedious to build dictionaries by hand.
- You can build dictionaries programmatically easily in two steps:
  1. Initialize a dictionary
     
     ```python
     birds = {}
     ```
  2. Update the dictionary keys.
     
     ```python
     birds['sparrow'] = 2
     if 'cardinal' in birds:
         birds['cardinal'] += 1
     else:
         birds['cardinal'] = 1
     ```
Counting words

Question: How do you count word frequencies in a text?

Some examples:
- Dictionaries: buildDict.py
- Default dictionaries: countWords.py
Advanced Topics

Various embeddings:

- Dictionaries of dictionaries
- Dictionaries of lists
- Lists of dictionaries

Things could get complicated, so you might want to switch to using a database at times