Dictionaries in Python

Behrang QasemiZadeh

me@atmykitchen.info

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).

```
>>> pos = {}
>>> pos
{}
>>> pos['colorless'] = 'ADJ'
>>> pos
{'colorless': 'ADJ'}
>>> pos['colorless']
'ADJ'
```

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).

```
>>> pos = {}
>>> pos
{}
>>> pos['colorless'] = 'ADJ'
>>> pos
{'colorless': 'ADJ'}
>>> pos['colorless']
'ADJ'
```

Define an empty dictionary

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).

```
>>> pos = {}
>>> pos
{}
>>> pos['colorless'] = 'ADJ'
>>> pos
{'colorless': 'ADJ'}
>>> pos['colorless']
'ADJ'
```

Assign to the key **“colorless”** the value of **“ADJ”**

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).

```
>>> pos = {}
>>> pos
{}
>>> pos['colorless'] = 'ADJ'
>>> pos
{'colorless': 'ADJ'}
>>> pos['colorless']
'ADJ'
```

Retrieve the assigned value to the key “colorless”

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
    - A look up tool (best exemplified by conventional dictionaries).

```
>>> pos = {}
>>> pos
{}
>>> pos['colorless'] = 'ADJ'
>>> pos
{'colorless': 'ADJ'}
>>> pos['colorless']
'ADJ'
>>> pos['green']
```

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).

```
>>> pos = {}
>>> pos
{}
>>> pos['colorless'] = 'ADJ'
>>> pos
{'colorless': 'ADJ'}
>>> pos['colorless']
'ADJ'
>>> pos['green']
```

```
Traceback (most recent call last):
      File "<stdin>", line 1, in ?
KeyError: 'green'
```

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).
  - Dictionaries are not sequences, thus the keys are not inherently ordered.

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).
  - Dictionaries are not sequences, thus the keys are not inherently ordered.

```
>>> list(pos)
['ideas', 'furiously', 'colorless', 'sleep']
>>> sorted(pos)
['colorless', 'furiously', 'ideas', 'sleep']
>>> [w for w in pos if w.endswith('s')]
['colorless', 'ideas']
```

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).
  - Dictionaries are not sequences, thus the keys are not inherently ordered.

```
>>> list(pos)
['ideas', 'furiously', 'colorless', 'sleep']
>>> sorted(pos)
['colorless', 'furiously', 'ideas', 'sleep']
>>> [w for w in pos if w.endswith('s')]
['colorless', 'ideas']
```

To find the keys, we can convert a dictionary to a list

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).
  - Dictionaries are not sequences, thus the keys are not inherently ordered.

```
>>> list(pos)
['ideas', 'furiously', 'colorless', 'sleep']
>>> sorted(pos)
['colorless', 'furiously', 'ideas', 'sleep']
>>> [w for w in pos if w.endswith('s')]
['colorless', 'ideas']
```

Dictionaries can be used in a context that a list can be used

Dictionaries in Python

- Dictionary data type can be seen as a tool for mapping between arbitrary data types:
  - A look up tool (best exemplified by conventional dictionaries).
  - Dictionaries are not sequences, thus the keys are not inherently ordered.

```
>>> list(pos)
['ideas', 'furiously', 'colorless', 'sleep']
>>> sorted(pos)
['colorless', 'furiously', 'ideas', 'sleep']
>>> [w for w in pos if w.endswith('s')]
['colorless', 'ideas']
```

Also, in loops!

Dictionaries: Main Methods

- **`keys`**(), **`values`**() and **`items`**() are methods to access dictionaries.

Dictionaries: Main Methods

- **`keys`**(), **`values`**() and **`items`**() are methods to access dictionaries.

```
>>> list(pos.keys())
['colorless', 'furiously', 'sleep', 'ideas']
>>> list(pos.values())
['ADJ', 'ADV', 'V', 'N']
>>> list(pos.items())
[('colorless', 'ADJ'), ('furiously', 'ADV'), ('sleep', 'V'), ('ideas', 'N')]
>>> for key, val in sorted(pos.items()):
        print(key + ":", val)

colorless: ADJ
furiously: ADV
ideas: N
sleep: V
```

Quiz:

- What is the output for the following code?

```
>>> pos['sleep'] = 'V'
>>> pos['sleep'] = 'N'
>>> pos['sleep']
```

Quiz:

- What is the output for the following code?

```
>>> pos['sleep'] = 'V'
>>> pos['sleep'] = 'N'
>>> pos['sleep']
'N'
```

Keys are unique: the key **’sleep’** is simply overwritten by the new value **‘N’**

Quiz:

- What is the output for the following code?

```
>>> pos['sleep'] = 'V'
>>> pos['sleep'] = 'N'
>>> pos['sleep']
'N'
```

To store multiple values for a key use a list value!

Quiz:

- What is the output for the following code?

```
>>> pos['sleep'] = ['V','N']
```

Defining Dictionaries

- Key-value pair format is used to create a dictionary:

```
>>> pos = {'colorless':'ADJ','ideas':'N','sleep':'V','furiously':'ADV'}
>>> pos = dict(colorless='ADJ', ideas='N', sleep='V', furiously='ADV')
```

- Keys must be immutable values (string, tuple, …) otherwise you get a **`TypeError:`**

```
>>> pos = {['ideas', 'blogs', 'adventures']: 'N'}
Traceback (most recent call last): File "<stdin>",
line 1, in <module>
TypeError: list objects are unhashable
```

Default Dictionaries

- If we try to access a key that is not in a dictionary, we get an error.
- We can use **`defaultdict`** to automatically create an entry for new keys and give them a default value

```
>>> from collections import defaultdict
>>> frequency = defaultdict(int)
>>> frequency['colorless'] = 4
>>> frequency['ideas']
0
>>> pos = defaultdict(list)
>>> pos['sleep'] = ['NOUN', 'VERB']
>>> pos['ideas']

[]
```

Default Dictionaries

- If we try to access a key that is not in a dictionary, we get an error.
- We can use **`defaultdict`** to automatically create an entry for new keys and give them a default value

```
>>> from collections import defaultdict
>>> frequency = defaultdict(int)
>>> frequency['colorless'] = 4
>>> frequency['ideas']
0
>>> pos = defaultdict(list)
>>> pos['sleep'] = ['NOUN'     VERB']
>>> pos['ideas']

[]
```

Check the size of **`pos`** dictionary to verify the functionality of the **`defaultdict(list)`**

Default Dictionaries

- If we try to access a key that is not in a dictionary, we get an error.
- We can use **`defaultdict`** to automatically create an entry for new keys and give them a default value

```
>>> from collections import defaultdict
>>> frequency = defaultdict(int)
>>> frequency['colorless'] = 4
>>> frequency['ideas']
0
>>> pos = defaultdict(list)
>>> pos['sleep'] = ['NOUN', 'VERB']
>>> pos['ideas']

[]
```

Functions list or int can be replaced by any other functions or expression, e.g. try

`defaultdict (lambda 'NOUN‘ )`

Default Dictionaries: Usage Example

- Sometimes, we like to work with a “fixed vocabulary”
- Let’s keep an inventory of top 100 frequent words in a corpus and replace the rest with special “out of vocabulary” token **UNK**:

```
>>> alice = nltk.corpus.gutenberg.words('carroll-alice.txt')
>>> vocab = nltk.FreqDist(alice)
>>> v100 = [word for (word, _) in vocab.most_common(100)]
>>> mapping = defaultdict(lambda: 'UNK')
>>> for v in v100:
      mapping[v] = v
>>> alice2 = [mapping[v] for v in alice]
>>> alice2[:10]
['UNK', 'Alice', "'", 's', 'UNK', 'in', 'UNK', 'by', 'UNK', 'UNK']
>>> len(set(alice2))
101
```

Incrementally Updating a Dictionary

- As practiced before, we can use dictionaries to count frequencies:

```
>>> from collections import defaultdict
>>> counts = defaultdict(int)
>>> from nltk.corpus import brown
>>> for (word, tag) in brown.tagged_words(\
      categories='news', tagset='universal'):
      counts[tag] += 1
>>> counts['NOUN']
30640
```

Incrementally Updating a Dictionary

- As practiced before, we can use dictionaries to count frequencies:

```
>>> from collections import defaultdict
>>> counts = defaultdict(int)
>>> from nltk.corpus import brown
>>> for (word, tag) in brown.tagged_words(\
      categories='news', tagset='universal'):
      counts[tag] += 1
>>> counts['NOUN']
30640
```

```
>>> sorted(counts)
['ADJ', 'PRT', 'ADV', 'X', 'CONJ', 'PRON', 'VERB', '.', 'NUM',
'NOUN', 'ADP', 'DET']
```

itemgetter to sort dictioanries by values

```
>>> from collections import defaultdict
>>> counts = defaultdict(int)
>>> from nltk.corpus import brown
>>> for (word, tag) in brown.tagged_words(\
      categories='news', tagset='universal'):
      counts[tag] += 1
```

```
>>> sorted(counts)
['ADJ', 'PRT', 'ADV', 'X', 'CONJ', 'PRON', 'VERB', '.', 'NUM', 'NOUN', 'ADP', 'DET']
```

```
>>> from operator import itemgetter
>>> sorted(counts.items(), key=itemgetter(1), reverse=True)
[('NOUN', 30640), ('VERB', 14399), ('ADP', 12355), ('.', 11928), ...]
>>> [t for t, c in \
      sorted(counts.items(), key=itemgetter(1), reverse=True)]
['NOUN', 'VERB', 'ADP', '.', 'DET', 'ADJ', 'ADV', 'CONJ', 'PRON',
'PRT', 'NUM', 'X']
```

itemgetter to sort dictioanries by values

```
>>> from collections import defaultdict
>>> counts = defaultdict(int)
>>> from nltk.corpus import brown
>>> for (word, tag) in brown.tagged_words(\
      categories='news', tagset[illegible]rsal'):
      counts[tag] += 1
```

```
>>> sorted(counts)
['ADJ', 'PRT', 'ADV', 'X', 'CONJ', 'PR[illegible], [illegible]VERB', '.', 'NUM', 'NOUN', 'ADP', 'DET']
```

```
>>> from operator import itemgetter
>>> sorted(counts.items(), key=itemgetter(1), reverse=True)
[('NOUN', 30640), ('VERB', 14399), ('ADP', 12355), ('.', 11928), ...]
>>> [t for t, c in \
      sorted(counts.items(), key=itemgetter(1), reverse=True)]
['NOUN', 'VERB', 'ADP', '.', 'DET', 'ADJ', 'ADV', 'CONJ', 'PRON',
'PRT', 'NUM', 'X']
```

`itemgetter` to sort dictioanries by values

- The first parameter of **`sorted()`** is the items to sort, a list of tuples (POS tag, frequency).
- The second parameter specifies the sort key using a function **`itemgetter().`**
- The last parameter of **`sorted`**() specifies that the items should be returned in reverse order.

```
>>> from operator import itemgetter
>>> sorted(counts.items(), key=itemgetter(1), reverse=True)
[('NOUN', 30640), ('VERB', 14399), ('ADP', 12355), ('.', 11928), ...]
>>> [t for t, c in \
      sorted(counts.items(), key=itemgetter(1), reverse=True)]
['NOUN', 'VERB', 'ADP', '.', 'DET', 'ADJ', 'ADV', 'CONJ', 'PRON',
'PRT', 'NUM', 'X']
```

Quiz

- Create an anagram dictionary using the list of words available from **`nltk.corpus.words.words('en')`**.
- An anagram of a word (phrase, or sentence) is obtained by rearranging its letters: “Angel” is an anagram of “glean”.

```
>>> words = nltk.corpus.words.words('en')
>>> anagrams = defaultdict(list)
>>> for word in words:
      key = ''.join(sorted(word))
```

Quiz

- Create an anagram dictionary using the list of words available from **`nltk.corpus.words.words('en')`**.
- An anagram of a word (phrase, or sentence) is obtained by rearranging its letters: “Angel” is an anagram of “glean”.

```
>>> words = nltk.corpus.words.words('en')
>>> anagrams = defaultdict(list)
>>> for word in words:
      key = ''.join(sorted(word))
      anagrams[key].append(word)
>>> anagrams[‘aent']
[u'ante', u'etna', u'neat', u'taen', u'tane', u'tean']
```

Quiz

- Create an anagram dictionary using the list of words available from **`nltk.corpus.words.words('en').`**
- An anagram of a word (phrase, or sentence) is obtained
  rearranging its letters: "Angel" is an anagram of "glea

```
>>> words = nltk.corpus.words.words('en')
>>> anagrams = defaultdict(list)
>>> for word in words:
      key = ''.join(sorted(word))
      anagrams[key].append(word)
>>> anagrams['aent']
[u'ante', u'etna', u'neat', u'taen', u'tane', u'tean']
```

You can replace these lines with a single line using **`nltk.Index`**

Quiz

- Create an anagram dictionary using the list of words available from **`nltk.corpus.words.words('en')`**.
- An anagram of a word (phrase, or sentence) is obtained
rearranging its letters: “Angel” is an anagram of “glea

You can replace these lines with a single line using **`nltk.Index`**

```
anagrams = nltk.Index((''.join(sorted(w)), w) for w in words)
```

```
>>> anagrams[‘aent']
[u'ante', u'etna', u'neat', u'taen', u'tane', u'tean']
```

Complex Keys and Values

- Often we need to use dictionaries with complex keys and values.
- For instance, we may like to guess the PoS of a word, given the word itself, and the tag of the previous word.

Complex Keys and Values

- Often we need to use dictionaries with complex keys and values.
- For instance, we may like to guess the PoS of a word, **given the word itself** and **the tag of the previous word**.

```
>>> pos = defaultdict(lambda: defaultdict(int))
>>> brown_news_tagged = nltk.corpus.brown.tagged_words(\
      categories='news', tagset='universal')
>>> for ((w1, t1), (w2, t2)) in \
      nltk.bigrams(brown_news_tagged):
      pos[(t1, w2)][t2] += 1
>>> pos[('DET', 'right')]
defaultdict(<class 'int'>, {'ADJ': 11, 'NOUN': 5})
```

Complex Keys and Values

- Often we need to use dictionaries with complex keys and values.
- For instance, we may like to guess the PoS of a word, **given the word itself** and **the tag of the previous word**.

```
>>> pos = defaultdict(lambda: defaultdict(int))
>>> brown_news_tagged = nltk.corpus.brown.tagged_words(\
      categories='news', tagset='universal')
>>> for ((w1, t1), (w2, t2)) in \
      nltk.bigrams(brown_news_tagged):
      pos[(t1, w2)][t2] += 1
>>> pos[('DET', 'right')]
defaultdict(<class 'int'>, {'ADJ': 11, 'NOUN': 5})
```

The default value is a dictionary of default value **`int`**(), i.e. zero

Complex Keys and Values

- Often we need to use dictionaries with complex keys and values.
- For instance, we may like to guess the PoS of a word, **given the word itself** and **the tag of the previous word**.

```
>>> pos = defaultdict(lambda: defaultdict(int))
>>> brown_news_tagged = nltk.corpus.brown.tagged_words(\
      categories='news', tagset='universal')
>>> for ((w1, t1), (w2, t2)) in \
      nltk.bigrams(brown_news_tagged):
      pos[(t1, w2)][t2] += 1
>>> pos[('DET', 'right')]
defaultdict(<class 'int'>, {'ADJ': 11, 'NOUN': 5})
```

In fact, we iterated over the bigrams in the corpus

Complex Keys and Values

- Often we need to use dictionaries with complex keys and values.
- For instance, we may like to guess the PoS of a word, **given the word itself** and **the tag of the previous word**.

```
>>> pos = defaultdict(lambda: defaultdict(int))
>>> brown_news_tagged = nltk.corpus.brown.tagged_words(\
      categories='news', tagset='universal')
>>> for ((w1, t1), (w2, t2)) in \
      nltk.bigrams(brown_news_tagged):
      pos[(t1, w2)][t2] += 1
>>> pos[('DET', 'right')]
defaultdict(<class 'int'>, {'ADJ': 11, 'NOUN': 5})
```

And, here is the compound key that stores the word and its previous tag

Inverting a Dictioanry

- Dictionaries are efficient for look-ups using keys
- But, finding a key given a value, i.e. "reverse lookup", is slow and cumbersome.
- If reverse lookup is often used, then we need to create a dictionary that maps values to keys:
  - Be cautions of multiple values, i.e. use **`defaultdict(list)`** to store !
  - Alternately, use NLTK

Inverting a Dictioanry

- Dictionaries are efficient for look-ups using keys
- But, finding a key given a value, i.e. "reverse lookup", is slow and cumbersome.
- If reverse lookup is often used, then we need to create a dictionary that maps values to keys:
  - Be cautions of multiple values, i.e. use **`defaultdict(list)`** to store !
  - Alternately, use NLTK

```
>>> pos = {'colorless': 'ADJ', 'ideas': 'N', 'sleep': 'V', 'furiously': 'ADV'}
>>> pos2 = nltk.Index((value, key) for (key, value) in pos.items())
>>> pos2['ADV']
['peacefully', 'furiously']
```

Dictionaries: Summary of Methods

Example	Description
`d = {}`	create an empty dictionary and assign it to d
`d[key] = value`	assign a value to a given dictionary key
`d.keys()`	the list of keys of the dictionary
`list(d)`	the list of keys of the dictionary
`sorted(d)`	the keys of the dictionary, sorted
`key in d`	test whether a particular key is in the dictionary
`for key in d`	iterate over the keys of the dictionary
`d.values()`	the list of values in the dictionary
`dict([(k1,v1), (k2,v2), ...])`	create a dictionary from a list of key-value pairs
`d1.update(d2)`	add all items from d2 to d1
`defaultdict(int)`	a dictionary whose default value is zero