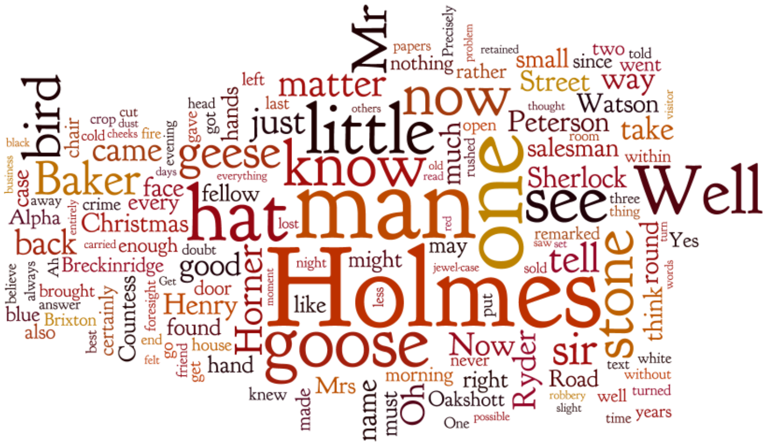
defaultdict tutorial

Michelle Fullwood

Motivation

Common task in text processing: doing a frequency count.



Common pattern

```
# frequency count for words in some text

text = "baa baa black sheep".split()

freqcount = dict()
for word in text:
    if word not in freqcount:
        freqcount[word] = 0
    freqcount[word] += 1

print freqcount

# {'sheep': 1, 'black': 1, 'baa': 2}
```

...made Pythonic

```
# frequency count for words in some text

text = "baa baa black sheep".split()

from collections import defaultdict
freqcount = defaultdict(int)
for word in text:
    freqcount[word] += 1

print freqcount
# defaultdict(<type 'int'>, {'sheep': 1, '
   black': 1, 'baa': 2})
```

How the source code would look

```
class defaultdict(dict):

  def __init__(self, default_factory=None, *a, **kw):
    if (default_factory is not None and
        not hasattr(default_factory, '__call__')):
      raise TypeError('first argument must be callable')
      dict.__init__(self, *a, **kw)
      self.default_factory = default_factory

  def __missing__(self, key):
    if self.default_factory is None:
      raise KeyError(key)
    self[key] = value = self.default_factory()
    return value

  def __getitem__(self, key):
    try:
      return dict.__getitem__(self, key)
    except KeyError:
      return self.__missing__(key)
```

(Credit: Jason Kirtland, ActiveState code Python recipes)

Behind the scenes

- `defaultdict` is a subclass of `dict`.
- Additional instance variable: `default_factory` (instantiated with `int` in our example).
- Additional function: `__missing__(key)`, returns `default_factory()`.

Behind the scenes

- When calling `defaultdict[key]`:
  - Call `dict.__getitem__(key)`. This returns the existing value if the key exists.
  - If the key doesn't exist in the dictionary, normally this raises a `KeyError`.
  - In a `defaultdict`, however, the function `__missing__(key)` is called instead.
  - This returns `default_factory()`, in our example `int()`, which is just 0.

Other possibilities for default_factory

	Default value	
`d = defaultdict(int)`	0	`d[key] += 1`
`d = defaultdict(list)`	[]	`d[key].append(listitem)`
`d = defaultdict(set)`	`set([])`	`d[key].add(setitem)`
`d = defaultdict(dict)`	{}	`d[key][secondkey] = val`

Exercises

1. Read in a text file

   (example: `https://sherlock-holm.es/stories/plain-text/blue.txt`)
2. Process each line word by word
3. Compile the following information:
   - A frequency count
   - The line numbers in which each word occurred.

     (If a word occurs multiple times in one line, don't collapse them.)
   - List of words occurring in the text, classified by length
4. Print out the following information:
   - The top 20 most frequent words.
   - The line numbers of the top 20 most frequent words.
   - The number of word types of each length

Beyond just types

`int`, `list`, `set` and `dict` are just functions that can take zero arguments.

`int() = 0, list() = [], set() = set([]), dict() = {}`.

If we supply `defaultdict` with a function `func` that takes no arguments, it will initialize any unseen key with `func()`!

Beyond just types

```
text = "baa baa black sheep".split()

def startatten():
    return 10

# frequency count for words in some text
from collections import defaultdict
freqcount = defaultdict(startatten)
for word in text:
    freqcount[word] += 1

print freqcount

# defaultdict(<function startatten at 0
   x127d140>, {'sheep': 11, 'black': 11, '
   baa': 12})
```

Using anonymous functions

Instead of defining `startatten`, we can also define an anonymous function using `lambda`.

`lambda: 10` is an anonymous function that does the same work as `startatten`.

`(lambda: 10)()` returns 10.

Beyond just types

```
text = "baa baa black sheep".split()

# frequency count for words in some text
# with add-one smoothing

from collections import defaultdict
freqcount = defaultdict(lambda: 1)
for word in text:
    freqcount[word] += 1

print freqcount


# defaultdict(<type 'int'>, {'sheep': 2, '
   black': 2, 'baa': 3})
```

Going deeper...

Defaultdict of a defaultdict

```
from collections import defaultdict

# count bigrams in text
bigram_count = defaultdict(lambda:
   defaultdict(int))

for word1, word2 in bigrams:
  bigram_count[word1][word2] += 1
```

Another variation

What if I want to make the default value dependent on the key?

Answer: subclass defaultdict

```
from collections import defaultdict

class ReflexiveDict(defaultdict):
    def __missing__(self, key):
        return key

mydict = ReflexiveDict(str)

mydict["baa"]
# "baa"
```

Going infinitely deep...

Infinitely-nested defaultdict

```
from collections import defaultdict

class recursivedefaultdict(defaultdict):
    def __init__(self):
        self.default_factory = type(self)

mydict = recursivedefaultdict(int):
mydict["To"]["infinity"]["and"]["beyond"]
```

Credit: Carsten Haese (comp.lang.python)

Thank you!

Resources used:

- wordle.org
- http://docs.python.org/2/library/collections.html
- http://code.activestate.com/recipes/523034-emulate-collectionsdefaultdict/
- http://www.itmaybeahack.com/homepage/books/nonprog/html/p10_set_map/p10_c04_defaultdict.html
- https://groups.google.com/forum/#!topic/comp.lang.python/lRnIhaJKZeo