Lightning Talk on defaultdict

Michelle Fullwood

Common pattern

```
# frequency count for words in some text

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

freqcount = dict()
for word in text:
    if word in freqcount:
        freqcount[word] += 1
    else:
        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})
```

Behind the scenes

Other possibilities

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

Beyond just types

`int`, `list` and `dict` are just functions that can take zero arguments. `int() = 0`, `list() = list[]`, `dict() = {}`.

If we supply `defaultdict` with a function `func` that takes no arguments, and 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
```

Using anonymous functions

Instead of defining `startatten`, we can also use anonymous functions.

These are equivalent:

```
def startatten: return 10
startatten = lambda: 10

def addtwo(n): return n+2
addtwo = lambda n: n+2
```

Beyond just types

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

# frequency count for words in some text
from collections import defaultdict
freqcount = defaultdict(lambda: 10)
for word in text:
    freqcount[word] += 1

print freqcount
```

Going deeper...

Defaultdict of a defaultdict

```
from collections import defaultdict

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

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

Going deeper

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)