Python Dictionary API

Here are some other methods that capture idioms often encountered with a dictionary `d`:

- `setdefault(key, default=None)` If `key` is in `d`, then return `d[key]`, otherwise insert `key` with value `default` and return `default`.
- `get(key, default=None)` If `key` is in `d`, then return `d[key]`, otherwise, return `default`.

Notice that `get` makes our counting function from the previous lecture more compact.

```
1 def wc_dict(tokens):
2     counts = {}
3     for token in tokens:
4         counts[token] = counts.get(token,0) + 1
5     return counts.items()
```

Practice

Suppose you have CSV file `phonebook.csv` with names and phone numbers of the form:

```
Donnie Wahlberg,123-555-1212
Jordan Knight,123-555-3456
Danny Wood,125-555-2311
Jonathan Knight,123-555-9999
Joey McIntyre,125-867-5309
```

Write a Python function `names_with_areacode(filename, areacode)` that returns a dictionary where each key is an area code and each value is a list of names of people who have phone numbers with that area code. For example:

```
>>> d = names_with_areacode('phonebook.csv', 123)
>>> d[125] = ["Joey McIntyre", "Danny Wood"]
>>> d[123] = ["Donnie Wahlberg", "Jordan Knight", "Jonathan Knight"]
```

JSON

JSON stands for JavaScript Object Notation. It is a data format and it has become a standard for lightweight data exchange on the web. Many websites that contain active updating use JSON to exchange data between the web server and the web browser. At its core, JSON is either a *dictionary*, a *list*, or a built-in *type* like string, integer, or boolean. It has several nice properties with respect to Python:

- JSON is valid Python;
- JSON is human readable;
- JSON is compact; and
- JSON is an accepted, ubiquitous format.

Here is an example bit of JSON representing a subset of a tweet it issued on 20 April 2015:

```
{'retweet_count': 3,
 'retweeted': False,
 'source': '<a href="http://sproutsocial.com" rel="nofollow">Sprout '
           'Social</a>',
 'text': 'Congratulations to Class of 1946 Environmental Fellow in Residence '
         '@ElizKolbert on her #Pulitzer for The Sixth Extinction!',
 'truncated': False,
 'user': {'contributors_enabled': False,
          'created_at': 'Tue Sep 16 15:13:15 +0000 2008',
          'default_profile': False,
          'default_profile_image': False,
          'description': 'Founded in 1793, Williams is a private, liberal '
                         'arts college located in Williamstown, Mass. '
                         'Tweets by the Office of Communications staff.',
          'entities': {'description': {'urls': []},
                       'url': {'urls': [{'display_url': 'williams.edu',
                                         'expanded_url': 'http://www.williams.edu',
                                         'indices': [0, 22],
                                         'url': 'http://t.co/55j6uDzJm5'}]}},
          'favourites_count': 569}}
```

Notice that you could cut and paste this JSON directly into python and you'd have a perfectly valid dictionary object. Doing this is usually not a good idea—it's better to use a parser—but Python has built-in library support for Python. Here are the primary functions:

Loading Data

If the file `williams.json` contains a single JSON object, then one can read it in using the `load` function.

```
1 import json
2
3 with open("williams.json") as fin:
4     d = json.load(fin)
```

One can use `loads` to load JSON data directly from a string.

```
>>> d = json.loads('{"Brent":40,"Courtney":41,"Oscar":6,"George":3}')
>>> d["George"]
3
```

Dumping Data

If `data` is a Python dictionary, list or built-in type like integer, string or boolean, then one can use either `json.dump` or `json.dumps` to save data to a file or to a string respectively. Here is an example that dumps a dictionary to a JSON string.

```
>>> d
{'brent': 40, 'Oscar': 6, 'Courtney': 41, 'George': 3}
>>> json.dumps(d)
'{"brent": 40, "Oscar": 6, "Courtney": 41, "George": 3}'
```

Here is an example that dumps that same dictionary to a file.

```
>>> json.dump(d,open("ages.json",'wt'))
$ cat ages.json
{"brent": 40, "Oscar": 6, "Courtney": 41, "George": 3}
```

Sets

Sets are like dictionaries without keys—you're interested in a collection of unique objects, but you're not interested in their order. Besides adding, removing, and testing membership, sets support the following set operations:

- union (elements appearing in at least one set),
- intersection (elements common to all sets), and
- symmetric difference.

By default these operation return new sets, but there are also version that are side-effecting.

```
>>> s = set(range(10))
>>> len(s)
10
>>> s.add(8)
>>> s
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> len(s)
10
>>> s2 = s.union(set(range(20)))
>>> s2
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19}
>>> s
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s3 = s2.intersection(set(range(10,20)))
>>> s3
{10, 11, 12, 13, 14, 15, 16, 17, 18, 19}
```