Python dictionaries and sets

Reuven M. Lerner, PhD
reuven@lerner.co.il

Dictionaries

- Key-value pairs
- “Hash tables”
- Extremely, extremely useful!
- If you don’t use these, then you aren’t “thinking in Python”

Dictionaries, continued

```
d = { }   # Empty dictionary
```

- Access/storage with [ ]
- Keys (indexes) — any hashable type
  - Strings, ints, and tuples are common
  - Not tuples containing lists!
- Values — any type at all

Working with dicts

```
d = {'a': 1, 'b': 2}

d['a']          # returns 1

d['a'] = 500    # sets value of 500

d.keys()        # ['a', 'b']

d.values()      # [500, 2]

del(d['a'])     # removes the pair
```

More efficient

- d.keys() and d.values() return lists!
- If your dictionary is quite large, this can use a lot of memory.
- You can, instead, use d.iterkeys() and d.itervalues()

Check keys with “in”

```
>>> d = {'a':1, 'b':2}

>>> 'a' in d

  True

>>> 'z' in d

  False
```

Or retrieve with a default:

```
>>> d =  {'a':1, 'b':2}

>>> d['z']

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

KeyError: 'z'

>>> d.get('z', 0)

  0
```

Remember

- Keys must be immutable types
- You can always use any type for keys and values (even mixing it up)
- Only one value per key
- Value can be another dict (or list, or tuple, or string, or ...)
- Dictionaries are unordered!

Looping on dicts

```
for k,v in d.items():

    print k

    print “{}: {}”.format(k, v)
```

- Worried that d.items() will return a large list? You can use d.iteritems(), which allocates one at a time

Looping on keys

```
for k in d:

    print k

    print “{}: {}”.format(k, d[k])
```

- Worried that d.items() will return a large list? You can use d.iteritems(), which allocates one at a time

What is d.items()?

```
d = {'a':1, 'b':2}

d.items()
  [('a', 1), ('b', 2)]

dict(d.items())
  {'a': 1, 'b': 2}
```

Updating dicts

- "update" lets you incorporate a dictionary into another dictionary
- This actually changes the updated dictionary!
- If an existing key appears in the argument, it is replaced/updated

```
>>> d = {'a':1, 'b':2}

>>> q = {'c':3, 'a':999}

>>> d.update(q)

>>> d

{'a': 999, 'b': 2, 'c': 3}
```

update and kwargs

- You can also invoke "update" with keyword arguments
- (You can do this with or without a dictionary parameter)

```
>>> d

{'a': 999, 'b': 2, 'c': 3}

>>> d.update(a=5)

>>> d

{'a': 5, 'b': 2, 'c': 3}
```

Update and lists

- You can also pass "update" a sequence of tuples

```
>>> d

{'a': 5, 'b': 2, 'c': 3}

>>> d.update([('z',26), ('y', 25)])

>>> d

{'a': 5, 'b': 2, 'c': 3, 'y': 25, 'z': 26}
```

Dicts and kwargs

- You can also create dictionaries like this:

```
d = dict(a=1, b=2)

d

    {'a': 1, 'b': 2}
```

Strings and dicts

```
d = {'a':1, 'b':2}
```

```
"a = %(a)s, b = %(b)03d" % d
```

```
'a = 1, b = 002'
```

Or use format

```
s = 'first name is {first}, last name is {last}'

s.format(first='Reuven', last='Lerner')
```

Sets

- We can use a dict as a set (using the keys, ignoring the values)
- Or we can use the built-in set object

```
s = set([1,2,3])

1 in s   # True

10 in s  # False
```

Sets

- Starting with Python 2.7, we can also say:

```
s = {1,2,3}
```

- The printed representation can be either:

```
set([1,2,3])    or    {1,2,3}
```

- And an empty set is still written as:

```
set()
```

Sets

- Because sets use hash(), all of the elements need to be hashable (basically, immutable with immutable contents)

```
s = set()

s.add([1,2,3])

  TypeError: unhashable type: 'list'
```

More with sets

```
s1 = {1,2,3}

s2 = {2,3,4}

# Return a new set:

s1.union(s2)            # s1 | s2, or {1,2,3,4}

s1.intersection(s2)     # s1 & s2, or {2,3}

# Change s1:

s1.add(100)             # s1 is now {1,2,3,100}

s1.remove(100)          # s1 is now {1,2,3} again
```

Mass updates

```
s1 = {1,2,3}
s2 = {2,3,4}

s1.update('abc')              # Pass a sequence
s1
  set(['a', 'b', 'c', 1, 2, 3])

s1.update(s2)                 # Pass a set
s1
  set(['a', 'b', 'c', 1, 2, 3, 4])
```

frozenset

- A frozenset is just like a set, except that it's immutable (and thus hashable)
- So, you cannot create a set of sets — but you can create a set of frozensets