Python Dictionaries

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

September 25, 2012

Outline

Outline

Dictionaries

Description

- Another useful data type built into Python is the dictionary.
- Dictionaries are sometimes found in other languages as "associative arrays".
- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type (strings and numbers can always be keys).
- Tuples can be used as keys if they contain only strings, numbers, or tuples;

Dictionaries

Description

- It is best to think of a dictionary as an unordered set of (key, value) pairs, with the requirement that the keys are unique (within one dictionary).
- A pair of braces creates an empty dictionary: {} (`dict` is the constructor).
- The main operations on a dictionary are storing a value with some key and extracting the value given the key.
- If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

Dictionaries

Example

```
>>> d = {}
>>> d, type(d)
({}, <type 'dict'>)
>>> d['Stephane'] = 7749
>>> d
{'Stephane': 7749}
>>> d['Mat'] = 7738
>>> d
{'Stephane': 7749, 'Mat': 7738}
>>> d = {'Stephane': 7749, 'Mat': 7738}
>>> d
{'Stephane': 7749, 'Mat': 7738}
>>> d['Guillaume']
Traceback (most recent call last):
KeyError: 'Guillaume'
>>>
```

Dictionaries

Example

```
>>> d = {'Stephane': 7749, 'Mat': 7738, 'Sylvain': 7741}
>>> d['Stephane'], d['Sylvain'], d['Mat']
(7749, 7741, 7738)
>>> 'Mat' in d     # 'Mat' is a key
True
>>> 'Guillaume' in d    # but 'Guillaume' is not
False
>>> len(d)
3
>>> d.keys()      # the keys
['Stephane', 'Mat', 'Sylvain']
>>> d.values()    # the values
[7749, 7738, 7741]
>>>
```

Dictionaries

Example

```
>>> # The dict() constructor builds dictionaries directly from
... # lists of key-value pairs stored as tuples.
...
>>> dict([('Stephane', 7749), ('Mat', 7738), ('Sylvain', 7741)])
{'Stephane': 7749, 'Mat': 7738, 'Sylvain': 7741}
>>>
>>> # When the keys are simple strings, it is sometimes easier
... # to specify pairs using keyword arguments
...
>>> dict(Stephane=7749, Mat=7738, Sylvain=7741)
{'Stephane': 7749, 'Sylvain': 7741, 'Mat': 7738}
>>>
>>> # Multiple keys with the values
>>> dict().fromkeys(('Stephane', 'Mat', 'Sylvain'), 1234)
{'Stephane': 1234, 'Mat': 1234, 'Sylvain': 1234}
>>>
```

Dictionaries

Example

```
>>> # Be careful
...
>>> d = dict()
>>> d.fromkeys(('Stephane', 'Mat', 'Sylvain'), 0000)
{'Stephane': 0, 'Mat': 0, 'Sylvain': 0}
>>> d
{}
>>>
>>> # Default value is None
...
>>> d.fromkeys(('Stephane', 'Mat', 'Sylvain'))
{'Stephane': None, 'Mat': None, 'Sylvain': None}
>>>
```

Outline

Dictionaries and values

Description

Values may have different types.

Example

```
>>> d = dict([('k1', 'a string'), ('k2', [1, 2]), ('k3', 123),
... ('k4', {})])
>>> d.keys()    # the list of all keys (no specific order)
['k3', 'k2', 'k1', 'k4']
>>> d.values()  # the list of all values (no specific order)
[123, [1, 2], 'a string', {}]
>>> ",".join([str(type(v)) for v in d.values()])
"<type 'int'>,<type 'list'>,<type 'str'>,<type 'dict'>"
>>>
```

Dictionaries

Example

```
>>> d = dict(Stephane=7749, Matthieu=7738, Sylvain=7741)
{'Stephane': 7749, 'Sylvain': 7741, 'Matthieu': 7738}
>>> # A first python idiom
...
>>> for k in d.keys():
...     print 'key=%s, value=%s' % (k, d[k])
...
key=Stephane, value=7749
key=Sylvain, value=7741
key=Matthieu, value=7738
>>> # Or (simpler)
>>> for k in d:
...     print 'key=%s, value=%s' % (k, d[k])
...
key=Stephane, value=7749
key=Sylvain, value=7741
key=Matthieu, value=7738
```

Modifying dictionaries

Example

```
>>> d = dict(Stephane=7749, Mat=7738, Sylvain=7741)
>>> # Modify one entry
...
>>> d['Stephane'] = 4977
>>> d['Stephane']
4977
>>> # Add one entry
...
>>> d['Arnaud'] = 7730
>>> d
{'Arnaud': 7730, 'Stephane': 4977, 'Sylvain': 7741, 'Mat': 7738}
>>> # Delete one entry
...
>>> del d['Stephane']
>>> d
{'Arnaud': 7730, 'Sylvain': 7741, 'Mat': 7738}
>>>
```

item

Example

```
>>> d = dict(Stephane=7749, Mat=7738, Sylvain=7741)
>>>
>>> d.items() # list of (key, value) pairs
[('Stephane', 7749), ('Sylvain', 7741), ('Mat', 7738)]
>>>
>>> ["%s=%s" % (k, v) for k, v in d.items()]
['Stephane=7749', 'Sylvain=7741', 'Mat=7738']
>>>
```

Outline

get

http://python.net/ goodger/projects/pycon/2007/idiomatic/handout.html

```
# We often have to initialize dictionary entries before use:

# This is the naive way to do it:
navs = {}
for (portfolio, equity, position) in data:
    if portfolio not in navs:
        navs[portfolio] = 0
    navs[portfolio] += position * prices[equity]

# dict.get(key, default) removes the need for the test
# D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None
navs = {}
for (portfolio, equity, position) in data:
    navs[portfolio] = (navs.get(portfolio, 0) + \
                       position * prices[equity])
```

setdefault

http://python.net/ goodger/projects/pycon/2007/idiomatic/handout.html

```
# Here we have to initialize mutable dictionary values.
# Each dictionary value will be a list.

# Initializing mutable dictionary values (the naive way)
equities = {}
for (portfolio, equity) in data:
    if portfolio in equities:
        equities[portfolio].append(equity)
    else:
        equities[portfolio] = [equity]
# dict.setdefault(key, default) does the job much more
# efficiently
# D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d
# if k not in D
equities = {}
for (portfolio, equity) in data:
    equities.setdefault(portfolio, []).append(equity)
```

setdefault

http://python.net/ goodger/projects/pycon/2007/idiomatic/handout.html

```
# setdefault can also be used as a stand-alone statement

navs = {}
for (portfolio, equity, position) in data:
    navs.setdefault(portfolio, 0)
    navs[portfolio] += position * prices[equity]

# The setdefault dictionary method returns the default value,
# but we ignore it here. We're taking advantage of setdefault's
# side effect, that it sets the dictionary value only if there
# is no value already.
```

zip is your friend

http://python.net/ goodger/projects/pycon/2007/idiomatic/handout.html

```
In [1]: given = ['John', 'Eric', 'Terry', 'Michael']

In [2]: family = ['Cleese', 'Idle', 'Gilliam', 'Palin']

In [3]: d = dict(zip(given, family))

In [4]: d
Out[4]: {'Eric': 'Idle', 'John': 'Cleese', 'Michael': 'Palin',
         'Terry': 'Gilliam'}

In [5]: d.keys()
Out[5]: ['John', 'Michael', 'Eric', 'Terry']

In [6]: d.values()
Out[6]: ['Cleese', 'Palin', 'Idle', 'Gilliam']
```

Removing duplicates

Example

```
In [1]: l = ['a', 'b', 'a', 'c', 'a', 'b', 'd', 'c']

In [2]: d = {}

In [3]: for x in l:
...:     if x not in d:
...:         d[x] = None
...:

In [4]: l = d.keys()

In [5]: l
Out[5]: ['a', 'c', 'b', 'd']

In [6]:
```

Removing duplicates

Example

```
In [1]: l = ['a', 'b', 'a', 'c', 'a', 'b', 'd', 'c']

In [2]: d = {}

In [3]: for x in l:
...:     d.setdefault(x, None)
...:

In [4]: l = d.keys()

In [5]: l
Out[5]: ['a', 'c', 'b', 'd']

In [6]:
```

Removing duplicates (without any dictionary)

Example

```
In [1]: l = ['a', 'b', 'a', 'c', 'a', 'b', 'd', 'c']

In [2]: # can't be shorter ...
...     l = list(set(l))

In [3]: l
Out[3]: ['a', 'c', 'b', 'd']

In [4]:
```