Dictionaries in Python

Pai H. Chou
National Tsing Hua University

Outline

- What is dictionary
- Create a dictionary
- Access (lookup value by key)
- Add or change content
- Membership test
- Methods and Built-in functions
- Applications

Dictionaries

- Unordered collection of key-value pairs
- Lookup syntax is similar to list
  - D[key], but key doesn't have to be int
  - Does not support slicing D[2:5]

```
>>> ec = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> ec['one']  # lookup the value by key
1
>>> ec['two']
2
>>> ec   # the key-value pairs may in different order
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

Restrictions

- keys: must be immutable!
  - similar to members of a set
- value unrestricted:
  - can be mutable or not
  - number, string, list, tuple, another dict

To create a dictionary

- use { } syntax

```
>>> d1 = { }   # make an empty dictionary
>>> d2 = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

- from list of key-value pairs

```
>>> d3 = dict([('one', 1), ('two', 2), ('three', 3), ('four', 4)])
>>> d3
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

- dictionary comprehension

```
>>> d4 = { i : 2**i for i in range(1,11)}
>>> d4  # maps i to 2**i
{1: 2, 2: 4, 3: 8, 4: 16, 5: 32, 6: 64, 7: 128, 8: 256, 9: 512, 10:
1024}
```

To lookup a dictionary

- use [ ] similar to list

```
>>> ec = {'one': 1, 'two': 2, 'three': 3}
>>> ec['two']
2
```

- What if the key is not in dictionary?

```
>>> ec['four']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'four'
```

- Solutions:
  - check key membership first before lookup
  - use .get() method

"safely" lookup

- key membership test

```
>>> ec = {'one': 1, 'two': 2, 'three': 3}
>>> 'four' in ec  # 'four' is not a key in ec
False
>>> 'two' in ec   # 'two' is a key in dictionary
True
```

- call the .get(k) method

```
>>> ec.get('two')   # same as ec['two'] regular lookup
2
>>> ec.get('four')  # this will return None if not found
>>> print(ec.get('four'))
None
```

  - if not defined, returns None

Dictionary methods for accessing content

- Getting content

`keys()`	returns the keys in dictionary (type: dict_keys)
`values()`	returns list of values in dictionary (type: dict_values)
`items()`	returns list of (key, value) pairs in dictionary
`get(`*k*`)`	look up *k*, return None if not in dictionary

```
>>> ec = {'one': 1, 'two': 2, 'three': 3}
>>> ec.keys()
dict_keys(['one', 'two', 'three'])
>>> ec.values()
dict_values([1, 2, 3])
>>> ec.items()
dict_items([('one', 1), ('two', 2), ('three', 3)])
```

Modifying a dictionary

- Add or modify an item (key-value pair)
  - just assign to it: *d*[*k*] = *v*
  - overwrites any existing key-value pair
- Delete an entry using **`del`** *d*[*k*]

```
>>> e2 = {'cat': '貓', 'dog': '狗', 'fish': '魚'}
>>> e2['chicken'] = '雞'  # add new key-value
>>> e2['dog'] = '犬'      # replaces the 'dog':'狗' entry!!!
>>> del e2['fish']        # delete entry whose key is 'fish'
>>> e2   # show current dictionary content
{'cat': '貓', 'dog': '犬', 'chicken': '雞'}
```

Dictionary methods

- Changing content

`update(`*d2*`)`	add dictionary d2's key-value pairs to this dictionary, overwriting any existing definitions of the same key
`clear()`	wipe out entire dictionary (make it empty content)

- Making Copy

`copy()`	make a shallow copy of the dictionary

Merging dictionaries

- *d*1.`update(`*d*2`)`
  - update *d*1 with items (*k*,*v*) from *d*2
  - add (*k*,*v*) if *k* not in *d*1; overwrite if *k* in *d*1.

```
>>> e1 = {'one': '一', 'dog': '狗'}
>>> e2 = {'two': '二', 'dog': '犬'}
>>> e1.update(e2)
>>> e1
{'one': '一', 'dog': '犬', 'two': '二'}
```

  - note: e1's original definition `'dog':'狗'` got replaced by new definition `'dog':'犬'` from e2

Copying and Clearing a dictionary

- dict's copy() method makes shallow copy
  - i.e., same number of key-value pairs as original
  - values reference the original ones, not copy
- dict's clear() method removes all entries

```
>>> e1 = {'one': '一', 'two': '二'}
>>> e2 = e1.copy()
>>> del e1['one']
>>> e1
{'two': '二'}
>>> e2                  # e2 is unaffected because it is a copy
{'one': '一', 'two': '二'}
>>> e2.clear()          # wipe out all entries
>>> e2
{}
```

Dictionary keys must be immutable

- Dictionaries: key-value pairs

```
>>> s = {'one': 1, 'two': 2, 'three': 3}  # initial dictionary
>>> s['four'] = 4; s['five'] = 5   # add/modify key-value pairs
>>> s['three']
3
>>> x = [1, 2, 3]          # make a list (which is mutable)
>>> s[x] = 123             # attempt to use mutable key
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Restrictions on keys

- Keys must be immutable
  - OK: numbers, strings, tuples
  - not OK: lists, dictionaries, sets, and other mutable
- Values can be any type, mutable or not

```
>>> d = {(1, 1): 'New Year', '(12, 25)': 'Xmas'}
>>> d[(8, 8)] = "Father's Day" # tuple can be key
>>> d[3.14159265] = 'pi'  # floating point number can be key
>>> d[()] = 'empty tuple' # () tuple is immutable, so it is ok as key
>>> d[()]
'empty tuple'
>>> d[[]] = 'empty list'  # [] list is mutable, so it is not ok as key
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Ordering of Entries

- The entries in dictionary are unordered
  - traditionally, the items can be in any order
  - Python 3.7: "insertion order"
    - previous version: can use OrderedDict class
  - but.. in general, you should not assume any order
- sort keys explicitly if you want ordering.

Example of sorting keys

- Python 3.7 or later, dictionary retains "insertion order"
  - overwriting does not change order
  - newly inserted keys are ordered at end

```
>>> e2 = {'fish': '魚', 'dog': '狗', 'cat': '貓'}
>>> e2.keys()   # keys are not sorted alphabetically
dict_keys(['fish', 'dog', 'cat']) # just in order of insertion (Py3.7)
>>> e2['dog'] = '犬'       # overwrite entry whose key is 'dog'
>>> e2.keys()              # overwriting doesn't change order of keys
dict_keys(['fish', 'dog', 'cat'])
>>> del e2['fish']         # delete entry whose key is 'fish'
>>> e2['chicken'] = '雞'   # insert a new one
>>> e2
{'dog': '犬', 'cat': '貓', 'chicken': '雞'}
```

%-style string formatting with dictionaries

- assumption: key is of string type
  - format string contains `%(`*key*`)s` for substitution
  - % *dictionary* as the argument, even if used multiple times

```
>>> D = {'lastname': 'Lee', 'firstname': 'Mary', 'phone': '123-4567'}
>>> print('name: %s %s, phone: %s' % (D['firstname'], D['lastname'], \
...   D['phone'])          # traditional way of formatting individually
name: Mary Lee, phone 123-4567
>>> print("name: %(firstname)s %(lastname)s, phone: %(phone)s" % D)
name: Mary Lee, phone: 123-4567
```