Dictionaries

Ruth Anderson

UW CSE 160

Winter 2016
Dictionaries or mappings

- A dictionary maps each *key* to a *value*
- Order does not matter
- Given a key, can look up a value
  - Given a value, cannot look up its key
- **No duplicate keys**
  - Two or more keys may map to the same value
- *Keys* and *values* are Python values
  - *Keys* must be immutable *(not a list, set, or dict)*
- Can add *key → value* mappings to a dictionary
  - Can also remove (less common)
Dictionary syntax in Python

```python
# Two different ways to create an empty dictionary

# Method 1: Using curly braces
d = { }  

d = dict()

# Method 2: Using the dict() constructor

us_wars_by_end = {
    1783: "Revolutionary",
    1848: "Mexican",
    1865: "Civil"  
}

us_wars_by_name = {
    "Civil": [1861, 1865],
    "Mexican": [1846, 1848],
    "Revolutionary": [1775, 1783]
}

• Syntax just like lists, for accessing and setting:

us_wars_by_end[1783]  ⇒
us_wars_by_end[1783][1:10]  ⇒
us_wars_by_name["WWI"] = [1917, 1918]
```

1783 → “Revolutionary”
1848 → “Mexican”
1865 → “Civil”

“Revolutionary” → 1775 1783
“Mexican” → 1846 1848
“Civil” → 1861 1865
Creating a dictionary

```python
>>> state_capitals = {"GA": "Atlanta", "WA": "Olympia" }

>>> phonebook = dict()
>>> phonebook["Alice"] = "206-555-4455"
>>> phonebook["Bob"] = "212-555-2211"

>>> atomic_number = {}
>>> atomic_number["H"] = 1
>>> atomic_number["Fe"] = 26
>>> atomic_number["Au"] = 79
```

"GA" → “Atlanta”
"WA" → “Olympia”

"Alice" → “206-555-4455”
“Bob” → “212-555-2211”

“H” → 1
“Fe” → 26
“Au” → 79
Accessing a dictionary

```python
>>> atomic_number = {"H":1, "Fe":26, "Au":79}
>>> atomic_number["Au"]
79
>>> atomic_number["B"]
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    atomic_number["B"]
KeyError: 'B'
>>> atomic_number.has_key("B")
False
>>> atomic_number.keys()
['H', 'Au', 'Fe']
>>> atomic_number.values()
[1, 79, 26]
>>> atomic_number.items()
[('H', 1), ('Au', 79), ('Fe', 26)]
```

Good for iteration (for loops)

```python
for key in mymap.keys():
    val = mymap[key]
    ... use key and val

for key in mymap:
    val = mymap[key]
    ... use key and val

for (key,val) in mymap.items():
    ... use key and val
```
Iterating through a dictionary

atomic_number = {"H":1, "Fe":26, "Au":79}

# Print out all the keys:
for element_name in atomic_number.keys():
    print element_name

# Another way to print out all the keys:
for element_name in atomic_number:
    print element_name

# Print out the keys and the values
for (element_name, element_number) in atomic_number.items():
    print "name:" , element_name, "number:" , element_number
Modifying a dictionary

```
us_wars1 = {
    "Revolutionary" : [1775, 1783],
    "Mexican" : [1846, 1848],
    "Civil" : [1861, 1865] }
```

```
us_wars1["WWI"] = [1917, 1918]  # add mapping

del us_wars_by_name["Mexican"]  # remove mapping
```
Dictionary Exercises

• What does this do?
  squares = { 1:1, 2:4, 3:9, 4:16 }
squares[3 + 3]
squares[2 + 2]

• Convert a list to a dictionary:
  • Given [5, 6, 7], produce {5:25, 6:36, 7:49}

• Reverse key with value in a dictionary:
  – Given {5:25, 6:36, 7:49}, produce {25:5, 36:6, 49:7}
Dictionary Exercise (Answers)

• Convert a list to a dictionary:
  – E.g. Given [5, 6, 7], produce {5:25, 6:36, 7:49}
    
    ```python
    d = {}
    for i in [5, 6, 7]:  # or range(5, 8)
        d[i] = i * i
    ```

• Reverse key with value in a dictionary:
  – E.g. Given {5:25, 6:36, 7:49}, produce {25:5, 36:6, 49:7}
    
    ```python
    k ={}
    for i in d.keys():
        k[d[i]] = i
    ```
A list is like a dictionary

• A list maps an integer to a value
  – The integers must be a continuous range 0..i

mylist = ['a', 'b', 'c']
mylist[1] ⇒ 'b'
mylist[3] = 'c'  # error!

• In what ways is a list more convenient than a dictionary?
• In what ways is a list less convenient than a dictionary?
Not every value is allowed to be a key in a dictionary

- Keys must be **immutable** values
  - int, float, bool, string, *tuple of immutable types*
  - *not*: list, set, dictionary

- The dictionary itself is **mutable** (e.g. we can add and remove elements)

- **Goal**: only dictionary operations change the keyset
  - after “mydict[x] = y”, mydict[x] ⇒ y
  - if a == b, then mydict[a] == mydict[b]
  These conditions should hold until mydict is changed

- **Mutable keys can violate these goals**