CS110 Week 10: Dictionaries & Sets

Prof. Jeff Johnson
based in part on Gaddis book

Lab 5: Strings, String Methods

- Due Thursday night!
- Hints:
  - 'A' == (chr(ord('a') – 32)) ➔ True
  - 'x' == (chr(ord('X') + 32)) ➔ True
  - 'B' > 'A' ➔ True
  - "dog" > "cat" ➔ True
  - "cat" > "car" ➔ True

Quiz 8

- Questions?

Dictionaries and Sets

- Our old familiar data types:
  - **int**eger: 1, 13, 42
  - **float**ing point: 3.1415, 0.25, 7502.34
  - **str**ing: "a", "abc", "Fred", "Now is the time\n"
  - **list**: [1, 2, 3], ["Fred", "John", "Mary", "Sue"]
  - **tuple**: (1,2,3), ("Fred", "John", "Mary", "Sue")
- Two more data-types
  - **dictionary**: {"Fred: 3", "Judy": 6, "Elaine: 9, "Mark": 2}
  - **set**: {1, 2, 3}, {"Fred", "Judy", "Elaine", "Mark"}

Dictionaries

- key : value pairs
  - engGer = {"one": "eins", "two": "zwei", "three": "drei"}
- Order is not important
  - Items not stored in order
- Reference values by keys, not index #
  - engGer["two"] ➔ "zwei"
  - engGer["four"] ➔ ***KeyError exception***

Dictionaries

- Keys must be *immutable* and *unique*
  - number, string, tuple (NOT list)
  - If duplicate keys, last one wins
- Values can be any data-type
  - number, string, list, tuple, …

Dictionary Examples

```
addrBook = {"Fred": ["Flintstone", "123 Foo St", "123-4567"],
        "Barney": ["Rubble", "456 Bar Ave", "987-6543"],
        "Wilma": ["Granite", "65 Baz Blvd", "654-3210"]  }

data = {"544-44-1234":
            ["Flintstone, Fred", "123 Foo St", "123-4567"],
        "123-45-5432":
            ["Rubble, Barney", "456 Bar Ave", "987-6543"],
        "987-65-4321":
            ["Granite, Wilma", "65 Baz Blvd", "654-3210"]  }
```

7

Working with Dictionaries

- ***in*** tests if key is in dictionary
  - e.g., John ***in*** addrBook #True if key "John" is there
  - Can use to guard against KeyError exceptions
  - ***not in*** tests if key is not in dictionary
- Get number of items:
  - len(<dictionary name>)

8

Dictionaries are Mutable

- Can create empty dictionary

```
myDictionary = {}
```

- Can add, delete, change items

```
engGer = {"one": "eins", "two": "zwei", "three": "drei"}
engGer["four"] = "veer"
engGer["four"] = "vier"

del engGer["four"]
```

9

Looping through Dictionary

for <key> in <dictionary>:
  do something with key or with
  <dictionary>[key]

Only 1 way to use ***while*** loop with dictionary

```
while len(d) > 0:
  item = d.popitem()
  <do something with item>
```

10

Dictionary Methods

- **<dictionary name>.<method>()**
- get(key): gets value for key (no KeyError)
- keys(): returns all keys
- values(): returns all values
- items(): returns all key:value pairs
- popitem(): returns random key:value pair and deletes it from dictionary
- clear(): empties the dictionary

11

Work on Lab 5

- Due Thursday night!
- Hints:
  - 'A' == (chr(ord('a') – 32)) ➔ True
  - 'x' == (chr(ord('X') + 32)) ➔ True
  - 'B' > 'A' ➔ True
  - "dog" > "cat" ➔ True
  - "cat" > "car" ➔ True

12

Sets

- Like mathematical sets
  - Items are unique
- Like Dictionaries, but with keys only
  - No key:value pairs; just keys

13

Creating Sets

- Create empty set

```
mySet = set()
```

- mySet = {<item1>, <item2>, <item3>, …}
- Or can use set() function with any sequence

```
mySet = set(<list>)          #duplicate items are discarded
mySet = set(<tuple>)
mySet = set(<string>)
mySet = set(<dictionary>)  #gives keys only
```

14

Working with Sets

- ***in*** tests if item is in a set
  - e.g., John ***in*** mySet #True if "John" is in set
  - ***not in*** tests if item is not in dictionary
- No way to index into sets
  - No need; just need to know if items in or not
- Get number of items:
  - len(<set name>)

15

Sets are Mutable (via methods)

- **<set name>.<method>()**
- add(<item>): adds item (unless duplicate)
- update(<sequence>): adds items
- remove(<item>): deletes item
- discard(<item>): deletes item (no KeyError)
- pop(): returns random item and deletes it
  - Note: not ***popitem()*** (which is for dictionaries)
- clear(): empties set

16

Looping through Set

```
for <item> in <set>:
  do something with item
```

Only 1 way to use ***while*** loop with set

```
while len(s) > 0:
  item = s.popitem()
  <do something with item>
```

17

Combining Sets

- set1 | set2: union
  - Or set1.union(set2)
- set1 & set2: intersection
  - Or set1.intersection(set2)
- set1 - set2: difference
  - Or set1.difference(set2)
- set1 ^ set2: symmetric difference
  - Or set1.symmetric_difference(set2)

18

Comparing Sets

- set1 > set2: superset?
- set1 >= set2: equal or superset?
  - Or set1.issubset(set2)
- set1 < set2: subset?
- set1 <= set2: equal or subset?
  - Or set1.issuperset(set2)

19

Reading for Next Week

- The next quiz will be about creating Web Apps using HTML. Please complete Sections 1-3 of HTML & CSS tutorial at: https://www.codecademy.com/learn/web
- The sections are:
  1. Intro to HTML
  2. HTML Structure: Using Lists
  3. HTML Structure: Tables, Divs, and Spans

20