Computer Science 134C
Introduction to Computer Science, in Python
Lecture #10 (Sets and Dictionaries)
September 28, 2018

Keywords
associations, `dict`, global variables, histogram, items, keys, memoization, `set`, values

We are introduced to all the main data types of python.

1. Questions?
2. From before: Lists.
3. From before: Tuples.
4. Sets. A `set` is a mutable, unordered collection of unique and immutable values. It models the mathematical concept.
   (a) Sets are created with `set()`. Given no arguments, the set is empty. Given an iterable, it populates the set with unique items encountered during iteration.
   (b) You can construct an *immutable* version of a set with the `frozenset` method:

   ```
   s = frozenset(set('The mountains, the mountains,'))
   ```

   constructs an immutable set, {`' '`,`','`,`'T'`,`'a'`,`'e'`,`'h'`,`'i'`,`'m'`,`'n'`,`'o'`,`'s'`,`'t'`,`'u'`}.
5. Set operations.
   (a) The size of a set is determined with `len`.
   (b) The empty set is `False`, all others are `True`.
   (c) You add elements to set with `add`. If the element is in the set already (using ==), it is not added again; the set holds unique values.
   (d) You remove a value from `s` with `s.discard(x)`.
   (e) The `s.pop()` method removes and returns an arbitrary value. It will fail if `s` is empty.
   (f) The `s.clear()` method removes all elements.
   (g) You can test for set membership with `in`.
   (h) You can compute the union of two sets with `s1 | s2` (or `s1.union(s2)`).
   (i) You can compute the intersection of two sets with `s1 & s2` (or `s1.intersection(s2)`).
   (j) You can compute the difference of two sets with `s1 - s2` (or `s1.difference(s2)`). The resulting set contains objects that are in `s1`, but not `s2`.
   (k) The symmetric difference is computed with `s1 ^ s2` (or `s1.symmetric_difference(s2)`). The resulting set contains objects that appear in exactly one of the two sets, `s1` or `s2`.
   (l) Comparison operations (==, <, <=, etc.) test for set equality and containment.
   (m) Sets are iterable. They can also be constructed using *comprehensions*:

   ```
   s = { c for c in 'the mountains' if c in 'aeiou' }
   ```

6. Dictionaries. A `dict` is an object that implements *mapping* or *association* from *keys* to `values`.

   (a) The `key` values must be immutable. The values may be (and typically are) mutable objects. Both keys and values may be mixed types (typically, they're not). The keys need not be comparable to each other; there is no natural order among the associations.

   (b) Dictionaries are created with `dict()`. Given no arguments, it creates an empty dictionary. It is important to avoid using `dict` as a variable name (this is hard!).

   (c) An association between key `k` and value `v` is entered in the dictionary `d` with `d[k] = v`. If `d` had a mapping from `k` before, it is replaced. Generally, association lookup looks like list indexing. However, dictionaries do not support slices.

   (d) `dict(iterable)`. Takes an iterable of key-value *pairs* (2-tuples) and uses the first elements as keys and the second elements as respective values.

   (e) `dict(key=value,...)`. Takes each keyword, turns it into a string, and inserts a mapping between the string and `value`.

   (f) You can test for key membership with `in`.

   (g) The *length* of the dictionary `d`, `len(d)`, is the number of associations it contains.

   (h) Iterating across a dictionary gives you an iterable over the keys. The `keys()` method gives you this same stream; because keys must be unique, there will be no repeated values. The `values()` method returns a stream of values that correspond, respectively, to the keys you would encounter; there may be duplicates, here. The `items()` method returns an iterable that delivers they key-value associations or (in Python-speak) *items*.

7. Methods of dictionaries.

   (a) `clear()`. Destructively removes all associations from the dictionary. Individual associations may be deleted with `del d[k]`. As with lists, `pop(k)` removes the association and returns the associated value.

   (b) `copy()`. Creates a new, shallow copy of the dictionary. The keys and values in the original and copy are shared references.

   (c) `d.get(k,x)`. Similar to `d[k] if k in d else x`.

$\star$