COMP10001 Foundations of Computing
Deictionaries and Sets

Semester 2, 2016

Chris Leckie

July 31, 2016

Announcements

- Tomorrow's lecture is on Project 1

Lecture Agenda

- Last lecture:
  - Functions, mutability, lists
- This lecture:
  - Dictionaries
  - Sets

So many numbers...

- While computers like numbers, we are not so numerically inclined. This can lead to mistakes in programming.
- Store the prices of drinks in a list: latte $3.50, chai $3.00, coke $2.

```
prices = [3.5, 3, 2]
```

- What's the cost of 478 lattes?

```
print(478 * prices[0]) # 0 is latte?
```

- We do not need a sequence here, we need a mapping from word to number

Dictionaries

- As powerful as sequences are, we sometimes want to be able to store/access items relative to a (unique) key, in which case we use a "dictionary" (aka lookup table, map, hash, index, associative array, ...):

Phone list:

Tim	41363
Andrew	41301
David	41302
Dengke	41303
Madison	41304
Jessica	41305

Favourite movies:

My Neighbour Totoro	4.5 stars
Once Were Warriors	5 stars
The Big Lebowski	6 stars
Kung Pow	5 stars
The Holy Grail	5 stars

- Should the keys of a dictionary be mutable?

Dictionaries: Basic Operations

- Dictionary initialisation:

```
mydict = {}
mydict = {"latte": 3.50, "chai": 3.00}
```

- Adding items to a dictionary:

```
dictionary[KEY] = VALUE
```

- Accessing items in a dictionary:

```
mydict[KEY]
mydict.get(KEY)  # no KeyError
```

- Deleting items from a dictionary (in-place):

```
a = dictionary.pop(KEY)
```

Basic Dictionary Manipulation

```
>>> drinks = {"chai": 3.50, "latte": 3.75}
>>> 3*drinks["chai"] + 2*drinks["latte"]
18.0
>>> drinks["chai"] = 4.00
>>> drinks
{'chai': 4.0, 'latte': 3.75}
>>> drinks["capuccino"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'capuccino'
>>> drinks.pop('latte')
3.75
>>> drinks
{'chai': 4.0}
```

More Dictionary Operations

- Key membership Test:

```
KEY in mydict # KEY found in mydict?
```

- Deleting the entire contents of a dictionary:

```
mydict.clear() # in-place
```

- Deleting an entry (no return value)

```
del mydict[KEY]
```

Removing items with del

```
>>> drink_dict = {'chai': 3.50, 'latte': 3.75}
>>> drink_list = ['chai', 'latte']
>>> drink_tuple = ('chai', 'latte')

>>> del drink_dict['chai']; print(drink_dict)
{'latte': 3.75}

>>> del drink_list[0]; print(drink_list)
['latte']

>>> del drink_tuple[0]; print(drink_tuple)
```

Dictionary views

```
mydict.keys()    # returns a 'view' of keys
mydict.values()  # returns view of values
mydict.items()   # returns (key,val) view
```

- a view will change if the underlying dictionary changes

```
>>> mydict = {'chai': 3.50, 'latte': 3.75}
>>> k = mydict.keys() ; print(k)
dict_keys(['chai', 'latte'])

>>> del drink_dict['chai']; print(k)
dict_keys(['latte'])
```

Class Exercise

Write a function that will return the phone number of a person that is supplied as a string argument to the function. You may assume that there is a dictionary called `phone_book` that has names as keys and phone numbers as values.
(No peeking at the next slide!)

Phone Book I

```
phone_book = {'andrew': 41312,
              'tim': 41334,
              'Beelzebubbles': 666}

def lookup(person):
    '''
    Return phone number for person
    Assumes phone_book is in this scope.
    '''
    if person in phone_book:
        return phone_book[person]

print(lookup('andrew'))
print(lookup('xxx'))
```

Phone Book II

```
phone_book = {'andrew': 41312,
              'tim': 41334,
              'Beelzebubbles': 666}

def lookup(person):
    '''
    Return phone number for person
    Assumes phone_book is in this scope.
    '''
    return phone_book.get(person)

print(lookup('andrew'))
print(lookup('xxx'))
```

Phone Book III

How would this be marked?

```
def lookup(person):                              2 marks
  '''
  Return phone number for person                 1 mark
  Assumes phone_book is in this scope.           1 mark
  '''
  return phone_book.get(person)                  3 marks
```

Sets I

- Sets are like lists but are unordered and elements are unique.

```
>>> a = {1,2,3,1,2,3,1,2,3}
>>> print(a)
{1,2,3}

>>> 1 in a    # testing for membership
True

>>> myset = set('abracadabra')
>>> print(myset)
{'a', 'r', 'b', 'c', 'd'}

a = set() # gotcha: {} is the empty dictionary
```

Sets II

```
>>> a = set('abra')
>>> b = set('alac')
>>> a                  # unique letters in a
{'r', 'a', 'b'}

>>> a - b              # set difference
{'b', 'r'}

>>> a | b              # set union
{'r', 'a', 'l', 'c', 'b'}

>>> a & b              # set intersection
{'a'}
```

Sets III

- Sets are not sequences (no order): you cannot slice or index them.
- BUT they are mutable with `add()` and `remove()`

```
>>> a = set('abra')
>>> a
{'r', 'a', 'b'}
>>> a.add('A') ; a
{'r', 'A', 'b', 'a'}
>>> a.add('A') ; a
{'r', 'A', 'b', 'a'}
>>> a.remove('a') ; a
{'r', 'A', 'b'}
>>> a.remove('a') ; a
```

Class Exercise

Given a list of words, print all of the unique words.
(Hmmm, this sounds familiar...)

Hashable

- Keys in dictionaries and elements in sets must be `hashable`
- All of Python's immutable built-in objects are hashable
  - `int`, `float`, `str`, `tuple`, `bool`
- Thus, `list`, `dict`, `set` cannot be keys in dictionaries or elements in sets

Lecture Summary

- What is a dictionary, and what basic operations can we perform on a dictionary?
- What types can be keys in a dictionary?
- What is a set?
- What types can be put into sets?
- Can you index (with an integer) or slice a dictionary or set? Why or why not?
- How do you create an empty dictionary and the empty set?
- How do you add and delete from dictionaries and sets?