Computer Science 1MD3
Tutorial 5: Data Types Continued

Aaron Simpson

Further to last tutorial, today you will be learning about several new data types usable in Python. The usefulness of some may not be apparent at this point, but for now all you need to understand is the syntax.

Review

As it's important to be very comfortable with the typing mechanisms involved with each of these data types it is helpful to have them memorized so that constant reference to syntax sheets is no longer necessary. To recapitulate:

- Lists
    - *listname = [element1,element2,elementn]*
    - *listname[n]* refers to the nth element in *listname*, with the first element having index 0
    - *listname[-n]* counts n elements backwards from the right, with the last element as index 1
    - *listname[-0]* is equivalent to *listname[0]*
    - *listname[0:2]* refers to a sublist composed of elements 0 and 1 (right side is non-inclusive)
    - *listname[1:]* refers to a sublist composed of element 1 and every element after 1
    - *listname[:1]* refers to a sublist composed of every element before 1 (but not including 1)
    - *listname[-2:]* refers to a sublist composed of the last two elements of the list
    - Index references involving a colon are known as “Slice Notation”
    - *len(listname)* will return the length of a list as an integer
    - *listname.index(element)* will return the index of the first element matching the one given
    - *listname.remove(element)* will delete the first instance of the given element from the list
    - *listname.append(element)* will concatenate the given element onto the end of the list
    - *listname.extend(list)* will break up a list and add each element onto the end of the first list
    - *listname.extend(element)* will give an error

- Tuples
    - *tuplename = (element1,element2,elementn)*
    - *tuplename[n]* refers to the nth element in *tuplename*, with the first element having index 0
    - *tuplename(n)* will return an error
    - Elements in a tuple cannot be altered
    - Tuples have no extend(), append(), or index() methods.
    - Slice Notation and negative indices also work when using tuples

Additional list operations

There are several methods for manipulating and analyzing lists in addition to index(), append(), and extend() which you have seen previously.

- *del listname[n]* will delete the element in index *n* from the list
- *listname.insert(i,x)* will insert object *x* immediately before index *i*
- *listname.pop([i])* will delete the element at list i and return its value as well
- *listname.count(x)* will return the number of times *x* appears in the list
- *listname.sort()* will sort the list in ascending order
- *listname.reverse()* will reverse the order of the elements in the list

Sets

Sets are somewhat like lists in that they are a collection of basic objects under a single name, however duplicates cannot exist in sets, and sets cannot be indexed as order is irrelevant. To generate a set, create an iterable object like a string or a list, and then call the set() function. This will not change the object, but will return a set based on it. You can assign a new object to contain this set. For example:

```
>>> mystring = “testing”
>>> myset = set(mystring)
>>> myset
```

Notice how duplicates are automatically removed and the order of the elements is automatically changed. Try making another object called *myset2* with a new set made out of the string *“another test”*. Once you have done that, print it to the screen, and then run the following tests:

```
>>> myset | myset2    #union
>>> myset & myset2    #intersection
>>> myset ^ myset2    #symmetric difference
>>> myset in myset2    #membership test
>>> 'e' in myset2
```

Dictionaries

In Python, a dictionary is a collection of values with no order that are accessed by “keys” rather than indices. Each dictionary key must be made out of a type with unchangeable elements. The syntax for defining a dictionary is as follows, notice the curly braces:

```
>>> d = {"key1":"value1", "key2":"value2", "keyn":"valuen"}
>>> d = dict( [ ("key1","value1"), ("key2","value2"), ("keyn","valuen") ] )
```

These two statements will both create the same dictionary.

Try entering this reference command:

```
>>> d["key2"]
```

Dictionaries have methods used to manipulate them like lists do.

- *d.keys()* will return a list of all of *d*'s keys
- *d.has_key(key)* will return *True* if *key* is in *d*'s list of keys, and will return *False* otherwise
- *d.iteritems()* will return an “iterator” over the dictionary's (key, value) pairs