- sequence
  - string
  - list
  - tuple
- set
- dict
A **sequence** is an iterable collection with random access.

- slicing: `seq[from:to+1]`

### Slicing

```python
a = ['spam', 'eggs', 100, 1234]

a[0]  # 'spam'
a[-2]  # 100
a[1:-1]  # ['eggs', 100]
a[:2]  # ['spam','eggs']
```
A list is a mutable sequence of objects.

Useful list methods

```python
da = [23, 42, 'foo']  # define list

da.append(x)         # append to list
a.pop(i)             # remove index i
a.sort()             # sort list
a.reverse()          # reverse list
...
```

See also

http://docs.python.org/2/tutorial/datastructures.html#more-on-lists
- `del` is convenient for deleting from lists

### Deleting from lists

- `del a[0]  # delete first element`
- `del a[2:4] # delete index 2 and 3`
- `del a[:] # clear list`
- A **tuple** is an immutable sequence of objects.
- defining a tuple is called 'tuple packing'

### Tuple packing

- \( t = (23, 42, \text{'foo'}) \)  #tuple (23,42,’foo’)
- \( t = 23, 42, \text{'foo'} \)  #tuple (23,42,’foo’)
- \( t = 23, (42, \text{'foo'}) \) #nested t. (23,(42,’foo’))
- \( t = () \)  #empty tuple
- \( t = 42, \)  #tuple (42)
sequence unpacking is the reverse operation to tuple packing

- tuple packing + seq. unpacking allows tight coding

### Sequence unpacking

```python
x, y, z = t  # t must be a sequence
# with 3 objects
```

### Tuple packing + sequence unpacking

```python
x, y, z = 1, 2, 3
a, b = b, a  # swap to variables
```
**Sequence looping**

```python
# loop over elements
for v in seq:
    print v

# loop over sorted elements
for v in sorted(seq):
    print v

# loop in reversed order
for v in reversed(seq):
    print v
```

- sorted and reversed do not change the original sequence!
Sequence looping + Tuple unpacking

```
x = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
# loop over elements
for a, b, c in x:
    print a + b + c
```
# Sequence looping

# loop over elements with indices
for i, v in enumerate(seq):
    print i, v

# looping two sequences at once
for x, y in zip(a, b):
    print x, y
A **set** is an unordered collection with no duplicate elements.

### Set operations

```
s = set([1,2,1,2,3])  # elements: [1,2,3]
s = {5, 4, 2}  # alternative syntax
```

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`a</td>
<td>b`</td>
</tr>
<tr>
<td><code>a &amp; b</code></td>
<td>intersect: elem. a and b</td>
</tr>
<tr>
<td><code>a - b</code></td>
<td>difference: elem. in a but not in b</td>
</tr>
<tr>
<td><code>a ^ b</code></td>
<td>complement: elem. a or b but not both</td>
</tr>
</tbody>
</table>

11/16
Set looping

#loop over elements
for v in set:
    print v

#loop over sorted elements
for v in sorted(set):
    print v
A **dict** is a key-value mapping.
- any immutable object can be key

### Definition and random access

```
d = {'foo': 42, 23: 'bar'}  # key: 'foo', val: 42
   # key: 23, val: 'bar'

d = dict(foo=42, bar=23)    # works only with 
   # string keys

d['foo']                   # 42
```
Dict looping

# Loop over keys and random access to values
for k in d.keys():
    print k, d[k]

# d.keys() is not necessary...
for k in d:
    print k, d[k]

# Loop over key-value pairs
for k, v in d.items():
    print k, v
Collection functions

# len and in are defined for all collections
len([1,2,3]) #=> 3
len(set((1,2))) #=> 2

5 in (1,2,3,4,5) #=> True
'c' not in dict(a=1, b=2, c=3) #=> False
9 in [1,2,3] #=> False

# del is defined for all mutable collections
d = dict(a=1, b=2)
del d['a']
Take home messages

- **list**: mutable sequence
- **tuple**: immutable sequence
- **set**: mathematical set operations
- **dict**: map
- looping: `for v in a`