## Sequence Types

- list, tuple, str; buffer, xrange, unicode

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \text{ in } s )</td>
<td>True if an item of ( s ) is equal to ( x ), else False</td>
</tr>
<tr>
<td>( x \text{ not in } s )</td>
<td>False if an item of ( s ) is equal to ( x ), else True</td>
</tr>
<tr>
<td>( s + t )</td>
<td>the concatenation of ( s ) and ( t )</td>
</tr>
<tr>
<td>( s * n, n * s )</td>
<td>( n ) shallow copies of ( s ) concatenated</td>
</tr>
<tr>
<td>( s[i] )</td>
<td>( i )'th item of ( s ), origin 0</td>
</tr>
<tr>
<td>( s[i:j] )</td>
<td>slice of ( s ) from ( i ) to ( j )</td>
</tr>
<tr>
<td>( s[i:j:k] )</td>
<td>slice of ( s ) from ( i ) to ( j ) with step ( k )</td>
</tr>
<tr>
<td>( \text{len}(s) )</td>
<td>length of ( s )</td>
</tr>
<tr>
<td>( \text{min}(s) )</td>
<td>smallest item of ( s )</td>
</tr>
<tr>
<td>( \text{max}(s) )</td>
<td>largest item of ( s )</td>
</tr>
</tbody>
</table>

```python
>>> lists = [[[[]]]] * 3
>>> lists
[['[]', '[]', '[]']]
```
>>> [1, 2] * 3
[1, 2, 1, 2, 1, 2]

>>> [] * 3
[]

>>> [[]] * 3
[[], [], []]

>>> a = [3]
>>> b = a * 3
>>> b
[3, 3, 3]

>>> a[0] = 4
>>> b
[4, 4, 4]

>>> a = [[3]]
>>> b = a * 3
>>> b
[[3], [3], [3]]

>>> a[0][0] = 4
>>> b
[[4], [4], [4]]

>>> a[0] = 5
>>> b
[[4], [4], [4]]

>>> a[0] += [2]
>>> b
[[4, 2], 5, [4, 2]]
# Pythonic Styles

- **do not write ...**
  
  ```python
  for key in d.keys():
  if d.has_key(key):
  i = 0
  for x in a:
      ...
      i += 1
  a[0:len(a) - i]
  for line in sys.stdin.readlines():
  for x in a:
      print x,
      print
  s = ""
  for i in range(lev):
      s += " "
  print s
  ```

- **when you can write ...**
  
  ```python
  for key in d:
  if key in d:
  for i, x in enumerate(a):
  a[:-i]
  for line in sys.stdin:
  for i, x in enumerate(a):
  print " ".join(map(str, a))
  print " " * lev
  ```
Tuples

immutable lists
Tuples and Equality

- caveat: singleton tuple
- `==`, `is`, `is not`

```python
>>> (1, 'a')
(1, 'a')
>>> (1)
1
>>> [1]
[1]
>>> (1,)
(1,)
>>> [1,]
[1]
>>> (5) + (6)
11
>>> (5,) + (6,)
(5, 6)
```

```python
>>> 1, 2 == 1, 2
(1, False, 2)
>>> (1, 2) == (1, 2)
True
>>> (1, 2) is (1, 2)
False
>>> "ab" is "ab"
True
>>> [1] is [1]
False
>>> 1 is 1
True
>>> True is True
True
```

```python
a += (1,2)  # new copy
a += [1,2]  # in-place
```
Comparison

- between the same type: “lexicographical”
- between different types: arbitrary
- `cmp()`: three-way `<`, `>`, `==`
- C: `strcmp(s, t)`, Java: `a.compareTo(b)`

```python
>>> (1, 'ab') < (1, 'ac')
True
>>> (1, ) < (1, 'ac')
True
>>> [1] < [1, 'ac']
True
>>> 1 < True
False
>>> True < 1
False
>>> [1] < [1, 2] < [1, 3]
True
>>> [1] == [1,] == [1.0]
True
>>> cmp ((1, ), (1, 2))
-1
>>> cmp ((1, ), (1, ))
0
>>> cmp ((1, 2), (1, ))
1
```
enumenrate

```python
>>> words = ['this', 'is', 'python']
>>> i = 0
>>> for word in words:
...    i += 1
...    print i, word
...
1 this
2 is
3 python

>>> for i, word in enumerate(words):
...    print i+1, word
...
```

- how to enumerate two lists/tuples simultaneously?
>>> a = [1, 2]
>>> b = ['a', 'b']

>>> zip (a,b)
[(1, 'a'), (2, 'b')]

>>> zip(a,b,a)
[(1, 'a', 1), (2, 'b', 2)]

>>> zip ([1], b)
[(1, 'a')]

>>> a = ['p', 'q']; b = [[2, 3], [5, 6]]
>>> for i, (x, [_, y]) in enumerate(zip(a, b)):
...     print i, x, y
...
0 p 3
1 q 6
zip and list comprehensions

```python
>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [(x, y) for x in vec1 for y in vec2]
[(2, 4), (2, 3), (2, -9), (4, 4), (4, 3), (4, -9), (6, 4), (6, 3), (6, -9)]

>>> [(vec1[i], vec2[i]) for i in range(len(vec1))]
[(2, 4), (4, 3), (6, -9)]

>>> sum([vec1[i]*vec2[i] for i in range(len(vec1))])
-34

>>> sum([x*y for (x,y) in zip(vec1, vec2)])
-34

>>> sum([v[0]*v[1] for v in zip(vec1, vec2)])
-34
```
how to implement zip?

binary zip: easy

```python
>>> def myzip(a,b):
...     if a == [] or b == []:
...         return []
...     return [(a[0], b[0])] + myzip(a[1:], b[1:]),
...     return [(a[0], b[0])] + myzip(a[1:], b[1:]),

>>> myzip([1,2], ['a','b'])
[(1, 'a'), (2, 'b')]
```

```python
>>> myzip([1,2], ['a','b'])
[(1, 'a'), (2, 'b')]
```

```python
>>> myzip([1,2], ['a','b'])
[(1, 'a'), (2, 'b')]
```

```python
>>> myzip([1,2], ['a','b'])
[(1, 'a'), (2, 'b')]
```

how to deal with arbitrarily many arguments?
Dictionaries

(heterogeneous) hash maps
Constructing Dicts

- key : value pairs

```python
>>> d = {'a': 1, 'b': 2, 'c': 1}
>>> d['b']
2
>>> d['b'] = 3
>>> d['b']
3
>>> d['e']
KeyError!
>>> d.has_key('a')
True
>>> 'a' in d
True
>>> d.keys()
['a', 'c', 'b']
>>> d.values()
[1, 1, 3]
```
Other Constructions

- zipping, list comprehension, keyword argument
- dump to a list of tuples

```python
>>> d = {'a': 1, 'b': 2, 'c': 1}
>>> keys = ['b', 'c', 'a']
>>> values = [2, 1, 1]
>>> e = dict(zip(keys, values))
>>> d == e
True
>>> d.items()
[('a', 1), ('c', 1), ('b', 2)]

>>> f = dict([(x, x**2) for x in values])
>>> f
{1: 1, 2: 4}

>>> g = dict(a=1, b=2, c=1)
>>> g == d
True
```
default values

- counting frequencies

```python
>>> def incr(d, key):
...     if key not in d:
...         d[key] = 1
...     else:
...         d[key] += 1
...

>>> def incr(d, key):
...     d[key] = d.get(key, 0) + 1
...

>>> incr(d, 'z')
>>> d
{'a': 1, 'c': 1, 'b': 2, 'z': 1}
>>> incr(d, 'b')
>>> d
{'a': 1, 'c': 1, 'b': 3, 'z': 1}
```
• best feature introduced in Python 2.5

```python
>>> from collections import defaultdict
>>> d = defaultdict(int)
>>> d['a']
0
>>> d['b'] += 1
>>> d
{'a': 0, 'b': 1}

>>> d = defaultdict(list)
>>> d['b'] += [1]
>>> d
{'b': [1]}

>>> d = defaultdict(lambda : <expr>)
```
# Mapping Type

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(a)</code></td>
<td>the number of items in <code>a</code></td>
</tr>
<tr>
<td><code>a[k]</code></td>
<td>the item of <code>a</code> with key <code>k</code></td>
</tr>
<tr>
<td><code>a[k] = v</code></td>
<td>set <code>a[k]</code> to <code>v</code></td>
</tr>
<tr>
<td><code>del a[k]</code></td>
<td>remove <code>a[k]</code> from <code>a</code></td>
</tr>
<tr>
<td><code>a.clear()</code></td>
<td>remove all items from <code>a</code></td>
</tr>
<tr>
<td><code>a.copy()</code></td>
<td>a (shallow) copy of <code>a</code></td>
</tr>
<tr>
<td><code>a.has_key(k)</code></td>
<td>True if <code>a</code> has a key <code>k</code>, else False</td>
</tr>
<tr>
<td><code>k in a</code></td>
<td>Equivalent to <code>a.has_key(k)</code></td>
</tr>
<tr>
<td><code>k not in a</code></td>
<td>Equivalent to <code>not a.has_key(k)</code></td>
</tr>
<tr>
<td><code>a.items()</code></td>
<td>a copy of <code>a</code>'s list of (key, value) pairs</td>
</tr>
<tr>
<td><code>a.values()</code></td>
<td>a copy of <code>a</code>'s list of values</td>
</tr>
<tr>
<td><code>a.get(k[, x])</code></td>
<td><code>a[k]</code> if <code>k</code> in <code>a</code>, else <code>x</code></td>
</tr>
<tr>
<td><code>a.setdefault(k[, x])</code></td>
<td><code>a[k]</code> if <code>k</code> in <code>a</code>, else <code>x</code> (also setting it)</td>
</tr>
<tr>
<td><code>a.pop(k[, x])</code></td>
<td><code>a[k]</code> if <code>k</code> in <code>a</code>, else <code>x</code> (and remove <code>k</code>)</td>
</tr>
</tbody>
</table>

*defaultdict behaves like setdefault, not get (following STL)*

[http://docs.python.org/lib/typesmapping.html](http://docs.python.org/lib/typesmapping.html)