Sequence Types

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

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n , n * s`	*n* shallow copies of *s* concatenated
`s[i]`	*i*'th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*

```
>>> lists = [[]] * 3
>>> lists
[[], [], []]
```

```
>>> lists[0].append(3)
>>> lists
[[3], [3], [3]]
```

the tricky *

```
>>> [1, 2] * 3
[1, 2, 1, 2, 1, 2]

>>> [] * 3
[]

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

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

>>> a[0] = 4
>>> b
[3, 3, 3]
```

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

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

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

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

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

>>> b[1] = 5
>>> b
[[4], 5, [4]]

>>> b[0] += [2]
[[4, 2], 5, [4, 2]]

>>> " " * 3
"   "

>>> "_ " * 3
"_ _ _ "
```

Pythonic Styles

- do not write ... when you can write ...

`for key in d.keys():`	`for key in d:`
`if d.has_key(key):`	`if key in d:`

`i = 0`
`for x in a:`
`    ...`
`    i += 1`
	`for i, x in enumerate(a):`

`a[0:len(a) - i]`	`a[:-i]`
`for line in \` `    sys.stdin.readlines():`	`for line in sys.stdin:`

`for x in a:`
`    print x,`
`print`
	`print " ".join(map(str, a))`

`s = ""`
`for i in range(lev):`
`    s += " "`
`print s`
	`print " " * lev`

Tuples

immutable lists

Tuples and Equality

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

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

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

```
>>> 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
```

Comparison

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

```
>>> (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
```

enumerate

```
>>> 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?

zip and _

```
>>> 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

```
>>> 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

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

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

how to deal with arbitrarily many arguments?

Dictionaries

(heterogeneous) hash maps

Constructing Dicts

- key : value pairs

```
>>> 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

```
>>> 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

```
>>> 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}
```

defaultdict

- best feature introduced in Python 2.5

```
>>> 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

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

defaultdict behaves like setdefault, not get (following STL)

http://docs.python.org/lib/typesmapping.html