String (immutable, ordered, characters)

```
s = 'string'.upper()          # STRING
s = 'fred'+'was'+'here'       # concatenation
s = ''.join(['fred', 'was', 'here']) # ditto
s = 'spam' * 3                # replication
s = str(x)                    # conversion
```

String iteration and sub-string searching

```
for character in 'str':        # iteration
    print (ord(character))     # 115 116 114
for index, character in enumerate('str')
    print (index, character)
if 'red' in 'Fred':            # searching
    print ('Fred is red')      # it prints!
```

String methods (not a complete list)

capitalize, center, count, decode, encode, endswith, expandtabs, find, format, index, isalnum, isalpha, isdigit, islower, isspace, istitle, isupper, join, ljust, lower, lstrip, partition, replace, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper, zfill

String constants (not a complete list)

```
from string import *           # I'm bad ...
print ((digits, hexdigits, letters,
         lowercase, uppercase, punctuation))
```

Old school string formatting (using % oper)

```
print ("It %s %d times" % ['occurred', 5])
# prints: 'It occurred 5 times'
```

Code	Meaning
s	String or string conversion
c	Character
d	Signed decimal integer
u	Unsigned decimal integer
H or h	Hex integer (upper or lower case)
f	Floating point
E or e	Exponent (upper or lower case E)
G or g	The shorter of e and f (u/l case)
%	Literal '%'

```
'%s' % math.pi      # --> '3.14159265359'
'%f' % math.pi      # --> '3.141593'
'%.2f' % math.pi    # --> '3.14'
'%.2e' % 3000       # --> '3.00e+03'
'%03d' % 5          # --> '005'
```

New string formatting (using format method)

Uses: 'template-string'.format(arguments)
Examples (using similar codes as above):

```
'Hello {}'.format('World')# 'Hello World'
'{}'.format(math.pi)      # ' 3.14159265359'
'{0:.2f}'.format(math.pi) # '3.14'
'{0:+.2f}'.format(5)      # '+5.00'
'{:.2e}'.format(3000)     # '3.00e+03'
'{:0>2d}'.format(5)       # '05' (left pad)
'{:x<3d}'.format(5)       # '5xx' (rt. pad)
'{:,}'.format(1000000)    # '1,000,000'
'{:.1%}'.format(0.25)     # '25.0%'
'{0}{1}'.format('a', 'b') # 'ab'
'{1}{0}'.format('a', 'b') # 'ba'
'{num:}'.format(num=7)    # '7' (named args)
```

List (mutable, indexed, ordered container)

Indexed from zero to length-1

```
a = []                       # the empty list
a = ['dog', 'cat', 'bird']   # simple list
a = [[1, 2], ['a', 'b']]     # nested lists
a = [1, 2, 3] + [4, 5, 6]    # concatenation
a = [1, 2, 3] * 456          # replication
a = list(x)                  # conversion
```

List comprehensions (can be nested)

Comprehensions: a tight way of creating lists

```
t3 = [x*3 for x in [5, 6, 7]] # [15, 18, 21]
z = [complex(x, y) for x in range(0, 4, 1)
          for y in range(4, 0, -1) if x > y]
# z --> [(2+1j), (3+2j), (3+1j)]
```

Iterating lists

```
L = ['dog', 'cat', 'turtle']
for item in L
    print (item)
for index, item in enumerate(L):
    print (index, item)
```

Searching lists

```
L = ['dog', 'cat', 'turtle']; value = 'cat'
if value in L:
    count = L.count(value)
    first_occurrence = L.index(value)
if value not in L:
    print 'list is missing {}'.format(value)
```

List methods (not a complete list)

Method	What it does
l.append(x)	Add x to end of list
l.extend(other)	Append items from other
l.insert(pos, x)	Insert x at position
del l[pos]	Delete item at pos
l.remove(x)	Remove first occurrence of x; An error if no x
l.pop([pos])	Remove last item from list (or item from pos); An error if empty list
l.index(x)	Get index of first occurrence of x; An error if x not found
l.count(x)	Count the number of times x is found in the list
l.sort()	In place list sort
l.reverse(x)	In place list reversal

List slicing

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8] # play data
x[2]     # 3rd element - reference not slice
x[1:3]   # 2nd to 3rd element (1, 2)
x[:3]    # The first three elements (0,1,2)
x[-3:]   # last three elements
x[:-3]   # all but the last three elements
x[:]     # every element of x – copies x
x[1:-1]  # all but first and last element
x[::3]  # (0, 3, 6, 9, …) 1st then every 3rd
x[1:5:2]# (1,3) start 1, stop >= 5, every2nd
```

Note: All Python sequence types support the above index slicing (strings, lists, tuples, bytearrays, buffers, and xrange objects)