The list class is one of the most useful in Python.

Both strings and lists are sequence types in Python, so share many similar methods. Unlike strings, lists are **mutable**.

If you change a list, it doesn’t create a new copy; it changes the input list.

Suppose you have 30 different test grades to average. You could use 30 variables: grade1, grade2, ..., grade30. Or you could use one list with 30 elements: grades[0], grades[1], ..., grades[29].

In file `AverageScores.py`:

```python
grades = [ 67, 82, 56, 84, 66, 77, 64, 64, 85, 67, 
           73, 63, 98, 74, 81, 67, 93, 77, 97, 65, 
           77, 91, 91, 74, 93, 56, 96, 90, 91, 99 ]
sum = 0
for score in grades:
    sum += score
average = sum / len(grades)
print(“Class average:”, format(average, ”.2f”))
```

```
> python AverageScores.py
Class average: 78.60
```
Creating Lists

Lists can be created with the list class constructor or using special syntax.

```python
>>> list()  # create empty list, with constructor
[]
>>> list([1, 2, 3])  # create list [1, 2, 3]
[1, 2, 3]
>>> list(['red', 3, 2.5])  # create heterogeneous list
['red', 3, 2.5]
>>> list(range(4))  # not an actual list
range(0, 4)
```

Many programming languages have an array type.

Arrays are:
- homogeneous (all elements are of the same type)
- fixed size
- permit very fast access time

Python lists are:
- heterogeneous (can contain elements of different types)
- variable size
- permit fast access time

Sequence Operations

Like strings, lists are sequences and inherit various functions from sequences.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x in s</td>
<td>x is in sequence s</td>
</tr>
<tr>
<td>x not in s</td>
<td>x is not in sequence s</td>
</tr>
<tr>
<td>s1 + s2</td>
<td>concatenates two sequences</td>
</tr>
<tr>
<td>s * n</td>
<td>repeat sequence s n times</td>
</tr>
<tr>
<td>s[i]</td>
<td>ith element of sequence (0-based)</td>
</tr>
<tr>
<td>s[1:j]</td>
<td>slice of sequence s from i to j-1</td>
</tr>
<tr>
<td>len(s)</td>
<td>number of elements in s</td>
</tr>
<tr>
<td>min(s)</td>
<td>minimum element of s</td>
</tr>
<tr>
<td>max(s)</td>
<td>maximum element of s</td>
</tr>
<tr>
<td>sum(s)</td>
<td>sum of elements in s</td>
</tr>
<tr>
<td>for loop</td>
<td>traverse elements of sequence</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>compares two sequences</td>
</tr>
<tr>
<td>==, !=</td>
<td>compares two sequences</td>
</tr>
</tbody>
</table>

Calling Functions on Lists

```python
>>> l1 = [1, 2, 3, 4, 5]
>>> len(l1)
5
>>> min(l1)  # assumes elements are comparable
1
>>> max(l1)  # assumes elements are comparable
5
>>> sum(l1)  # assumes summing makes sense
15
>>> l2 = [1, 2, "red"]
>>> sum(l2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> min(l2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
```
Aside: Functions vs. Methods

Since lists are actual objects in class `lst`, shouldn’t `len`, `max`, etc. be `methods` instead of functions? Yes and no!

Remember from earlier that `len` is actually syntactic sugar for the method `_len_`.

```python
>>> len([1, 2, 3])
3
>>> [1, 2, 3]._len_()
3
```

The others (`sum`, `max`, `min`) are actually functions defined on the class, for user convenience.

You just have to remember which operators are functions and which are methods.

Using Functions

We could rewrite `AverageScores.py` as follows:

```python
grades = [67, 82, 56, 84, 77, 64, 64, 85, 67, 73, 63, 98, 74, 81, 67, 93, 77, 97, 65, 77, 91, 91, 74, 93, 56, 96, 90, 91, 99]
average = sum(g) / len(g)
print("Class average: ", format(average, ".2f"))
```

```bash
> python AverageScores.py
Class average: 78.60
```

Comparing Lists

Compare lists using the operators: `>`, `>=`, `<`, `<=`, `==`, `!=`. Uses lexicographic ordering: Compare the first elements of the two lists; if they match, compare the second elements, and so on. The elements must be of comparable classes.

```python
>>> list1 = ["red", 3, "green"]
>>> list2 = ["red", 3, "grey"]
>>> list1 < list2
True
>>> list3 = ["red", 5, "green"]
>>> list3 > list1
True
>>> list4 = [5, "red", "green"]
>>> list3 < list4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
>>> ["red", 5, "green"] == [5, "red", "green"]
False
```

BTW: the book’s comparisons in 10.2.8 seem wrong.
List Comprehension

List comprehension gives a compact syntax for building lists.

```python
>>> range(4)  # not actually a list
range(0, 4)
>>> [x for x in range(4)]  # create list from range
[0, 1, 2, 3]
>>> [x ** 2 for x in range(4)]
[0, 1, 4, 9]
>>> lst = [2, 3, 5, 7, 11, 13]
>>> [x ** 3 for x in lst]
[8, 27, 125, 343, 1331, 2197]
>>> [x for x in lst if x > 2]
[3, 5, 7, 11, 13]
>>> [s[0] for s in ['red', 'green', 'blue'] if s <= 'green']
['g', 'b']
>>> from IsPrime3 import *
>>> [x for x in range(100) if isPrime(x)]
```

More List Methods

These are methods from class list. Since lists are mutable, these actually change l.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l.append(x)</td>
<td>add x to the end of l</td>
</tr>
<tr>
<td>l.extend(12)</td>
<td>append elements of [2] to l</td>
</tr>
<tr>
<td>l.insert(i, x)</td>
<td>insert x into l at position i</td>
</tr>
<tr>
<td>l.pop()</td>
<td>remove and return the last element of l</td>
</tr>
<tr>
<td>l.pop(i)</td>
<td>remove and return the ith element of l</td>
</tr>
<tr>
<td>l.remove(x)</td>
<td>remove the first occurrence of x from l</td>
</tr>
<tr>
<td>l.reverse()</td>
<td>reverse the elements of l</td>
</tr>
<tr>
<td>l.sort()</td>
<td>order the elements of l</td>
</tr>
<tr>
<td>l.count(x)</td>
<td>number of times x appears in l</td>
</tr>
<tr>
<td>l.index(x)</td>
<td>index of first occurrence of x in l</td>
</tr>
</tbody>
</table>

List Examples

```python
>>> l1 = [1, 2, 3]
>>> l1.append(4)  # add 4 to the end of l1
>>> l1
[1, 2, 3, 4]
>>> l1.count(4)   # count occurrences of 4 in l1
1
>>> l2 = [5, 6, 7]
>>> l1.extend(l2) # add elements of l2 to l1
>>> l1
[1, 2, 3, 4, 5, 6, 7]
>>> l1.index(5)   # where does 5 occur in l1?
4
>>> l1.insert(0, 0) # add 0 at the start of l1
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7]
>>> l1.insert(3, 'a') # note new value of l1
>>> l1
[0, 1, 2, 'a', 3, 4, 5, 6, 7]
>>> l1.remove('a') # what goes in can come out
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7]
```
### List Examples

```python
>>> l1.pop()  # remove and return last element
7
>>> l1
[0, 1, 2, 3, 4, 5, 6]
>>> l1.reverse()  # reverse order of elements
>>> l1
[6, 5, 4, 3, 2, 1, 0]
>>> l1.sort()  # elements must be comparable
>>> l1
[0, 1, 2, 3, 4, 5, 6]
>>> l2 = [4, 1.3, "dog"]
>>> l2.sort()  # elements must be comparable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'float'
>>> l2.pop()  # put the dog out
'dog'
>>> l2
[4, 1.3]
>>> l2.sort()  # int and float are comparable
>>> l2
[1.3, 4]
```

### Random Shuffle

A useful method on lists is `random.shuffle()` from the `random` module.

```python
>>> import random
>>> list1 = [x for x in range(9)]
>>> list1
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> random.shuffle(list1)
>>> list1
[7, 4, 0, 8, 1, 6, 5, 2, 3]
>>> random.shuffle(list1)
>>> list1
[4, 1, 5, 0, 7, 8, 3, 2, 6]
>>> random.shuffle(list1)
>>> list1
[7, 5, 2, 6, 0, 4, 3, 1, 8]
```

### Splitting a String into a List

Recall our `SplitFields` function from Slideset 8 to split up a comma separated value (csv) string. Python provides an easier approach with the `split` method on strings.

```python
>>> str1 = "abc, def, ghi"
>>> str1.split(",")  # split on comma
['abc', 'def', 'ghi']
>>> strs = " abc def ghi "
>>> strs.split()  # keep whitespace
['abc', 'def', 'ghi']
>>> str3 = "tabc\ndef\nghi"
>>> str3.split()  # split on whitespace
['abc', 'def', 'ghi']
>>> str4 = "abc / def / ghi"
>>> str4.split("/")  # split on slash
['abc', 'def', 'ghi']
```

Note split with no arguments splits on whitespace.

### Processing CSV Lines

Suppose grades for a class were stored in a list of csv strings, such as:

```python
studentData = ["Charlie,90,75",
              "Frank,8,77",
              "Susie,60,80"]
```

Here the fields are: Name, Midterm grade, Final Exam grade.

Compute the average for each student and print a nice table of results. *Remember that we solved a version of this problem in Slideset 3, where the data was entered by the user.*
def ProcessStudentData ( studentData ):
    """ Process list of csv student records. """
    # Print header line:
    print ( "Name MT FN Avg"
    print ( "-----------------------------"
    for line in studentData :
        fields = line . split (',')
        if ( len ( fields ) < 3):
            print ( "Bad student record for ", fields [0] )
            continue
        else :
            name , midterm , final = fields [0].strip () ,
            int ( fields [1].strip () ),
            int ( fields [2].strip () )
        avg = ( midterm + final ) / 2
        print ( format (name , "10s"), \n        format ( midterm , "4d"), \n        format ( final , "4d"), \n        format (avg , " 7.2 f") )

def main () :
    studentData = ["Charlie ,90 ,75",
                   "Frank ,8 ,77",
                   "Johnnie ,40",
                   "Susie ,60 ,80"]
    ProcessStudentData( studentData )

main ()

> python ExamExample2.py

Name MT FN Avg
-----------------------------
Charlie 90 75 82.50
Frank 8 77 42.50
Bad student record for Johnnie
Susie 60 80 70.00

Suppose you want to make a copy of a list. The following won’t work!

>>> lst1 = [1, 2, 3, 4]
>>> lst2 = lst1
>>> lst1 is lst2    # there’s only one list here
True
>>> print(lst1)
[1, 2, 3, 4]
>>> print(lst2)
[1, 2, 3, 4]
>>> lst1.append(5) # changes to lst1 also change lst2
>>> print(lst2)
[1, 2, 3, 4, 5]

But you can do the following:

>>> lst2 = [x for x in lst1] # creates a new copy

Like any other mutable object, when you pass a list to a function, you’re really passing a reference (pointer) to the object in memory.

def alter( lst ) :
    lst.pop()

def main () :
    lst = [1, 2, 3, 4]
    print ( "Before call: ", lst )
    alter( lst )
    print ( "After call: ", lst )

main ()

> python ListArg.py

Before call: [1, 2, 3, 4]
After call: [1, 2, 3]
In Slideset 7 we introduced the Card class. Let’s now define a Deck of Cards. Remember we defined some functions: isRank, isSuit, cardRankToIndex, cardIndexToRank, etc.

It would be much easier to just add the following constant definitions to Card.py.

```python
class RANKS = ['Ace', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King']
SUITS = ['Spades', 'Diamonds', 'Hearts', 'Clubs']
def isRank(r):
    return r in RANKS
def isSuit(s):
    return s in SUITS
def cardRankToIndex(r):
    return RANKS.index(r)
def cardSuitToIndex(s):
    return SUITS.index(s)
```

Think of how you’d redefine the functions listed above with those lists available.

A deck of cards “is” a list of Card objects, one for each combination of rank and suit.

**Data:** a list of Card objects, initially all possible combinations of rank and suit.

**Methods:**
- Print the deck in order.
- Shuffle the deck.
- Deal a card from deck.
- How many cards are left in the deck (after dealing)?
Create a Card Deck

In file Deck.py:

```python
import random
from Card import *

class Deck:
    """ Defines the Deck class. Each Deck contains a list of cards, one for each rank and suit ""
    def __init__(self):
        """ Return a new deck of cards. ""
        self.__cards = []
        for suit in Card.SUITS:
            for rank in Card.RANKS:
                c = Card(rank, suit)
                self.__cards.append(c)

    def shuffle(self):
        """ Shuffle the cards. ""
        random.shuffle(self.__cards)

    def deal(self):
        """ Remove and return the top card, or None if the deck is empty. ""
        if len(self) == 0:
            print("Deck is empty.")
            return None
        else:
            return self.__cards.pop(0)

    def __str__(self):
        result = ""
        for c in self.__cards:
            # Here we ask each card how it wants to be printed.
            result = result + str(c) + "\n"
        return result
```

Dealing a Card and Deck Length

Dealing a Card means removing the top card from the Deck and returning that card:

```python
def deal(self):
    """ Remove and return the top card, or None if the deck is empty. ""
    if len(self) == 0:
        print("Deck is empty.")
        return None
    else:
        return self.__cards.pop(0)
```

Notice that we're calling `len(self)` to check whether the Deck is empty. This only works if we define the `__len__` method for the class:

```python
def __len__(self):
    """ Returns the number of cards left in the deck. ""
    return len(self.__cards)
```

Printing a Deck

Finally, we can use the print method for Deck class instances only if we've defined a `__str__` method to generate an appropriate string value:

```python
def __str__(self):
    result = ""
    for c in self.__cards:
        result = result + str(c) + "\n"
    return result
```

Notice that `str(c)` only works because we defined the `__str__` method within class Card.

Card Deck Example

Other things we might want to do with a deck are:

- shuffle the deck
- deal a card from the deck
- ask how many cards are left in the deck
- print the deck in order

Since the deck "is" a list, shuffling just means calling the `random.shuffle` function.

```python
def shuffle(self):
    """ Shuffle the cards. ""
    random.shuffle(self.__cards)
```

Since lists are mutable, this shuffles in place, i.e., it doesn't create a new deck.
Using the Deck Class

```python
>>> from Deck import *
>>> d = Deck()  # create a new deck
>>> print(d)  # print, notice order
Ace of Spades
2 of Spades
...,
Jack of Clubs
Queen of Clubs
King of Clubs

>>> d.shuffle()  # randomly shuffle deck
>>> print(d)
Queen of Spades
5 of Diamonds
4 of Clubs
...
Jack of Diamonds
8 of Clubs
```

Designing the Hand Class

Recall that our initial goal (from the Object slideset) was playing Poker. Now that we have Cards and Decks, we can define Hands; a poker hand is five cards.

**Data:** a list of five Card objects, dealt from a Deck object.

**Methods:**
- Print the hand in order.
- (Later) evaluate the hand as a poker hand.

The Hand Class

From file Hand.py:

```python
import Card
from Deck import *

class Hand:
    ''' Five cards dealt from a Deck object. '''
    def __init__(self, deck):
        ''' A hand is simply a list of 5 cards, dealt from the deck. '''
        if (len(deck) < 5):
            print("Not enough cards left!")
            return None
        self.__cards = []
        for i in range(5):
            card = deck.deal()  # deal next card
            self.__cards.append(card)  # append to hand

    def __str__(self):
        result = ""
        for card in self.__cards:
            result = result + str(card) + 
        return result
```

```python
>>> c1 = d.deal()  # deal top card
>>> print(c1)
Queen of Spades

>>> c2 = d.deal()  # deal next card
>>> print(c2)
5 of Diamonds

>>> len(c1)  # didn't define len for Card
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'Card' has no len()

>>> len(d)  # deck now 50 cards
50

>>> d.__len__()  # len same as __len__
50

>>> d.__cards  # can't access private field
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Deck' object has no attribute '__cards'
```
Finally, we allow looking at the cards in the Hand object:

```python
def getCard(self, i):
    """ Get the ith card from the hand, where i in [0..4]. """
    if (0 <= i <= 4):
        return self.__cards[i]
    else:
        return None
```

Using the Hand Class

```python
>>> from Hand import *
>>> h1 = Hand()             # can't deal without a deck
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __init__() missing 1 required positional argument: 'deck'
>>> d = Deck()              # so create a new deck
>>> d.shuffle()             # shuffle it
>>> print(d)
7 of Clubs
King of Diamonds
6 of Diamonds
Queen of Spades
8 of Clubs
Jack of Hearts
8 of Hearts
...
7 of Spades
10 of Clubs
```

Future Work

It would be nice to be able to evaluate a hand as a poker hand, and perhaps compare two hands.

*That would be a pretty good project!*
Next stop: More on Lists.