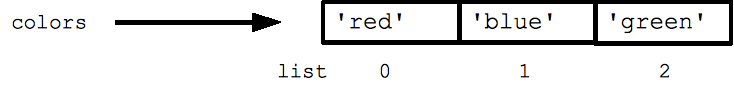
Python Lists

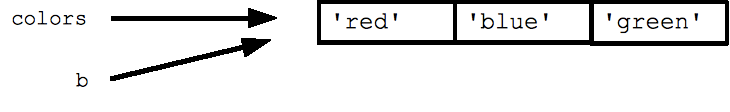
Python has a great built-in list type named "list". List literals are written within square brackets [ ]. Lists work similarly to strings -- use the len() function and square brackets [ ] to access data, with the first element at index 0.

  colors = ['red', 'blue', 'green']  
  print colors[0]    ## red  
  print colors[2]    ## green  
  print len(colors)  ## 3



Assignment with an = on lists does not make a copy. Instead, assignment makes the two variables point to the one list in memory.

  b = colors   ## Does not copy the list



The "empty list" is just an empty pair of brackets [ ]. The '+' works to append two lists, so [1, 2] + [3, 4] yields [1, 2, 3, 4] (this is just like + with strings).

FOR and IN

Python's \*for\* and \*in\* constructs are extremely useful, and the first use of them we'll see is with lists. The \*for\* construct -- for **var** in **list** -- is an easy way to look at each element in a list (or other collection). Do not add or remove from the list during iteration.

  squares = [1, 4, 9, 16]  
  sum = 0  
  for num in squares:  
    sum += num  
  print sum  ## 30

If you know what sort of thing is in the list, use a variable name in the loop that captures that information such as "num", or "name", or "url". Since python code does not have other syntax to remind you of types, your variable names are a key way for you to keep straight what is going on.

The \*in\* construct on its own is an easy way to test if an element appears in a list (or other collection) -- **value** in**collection** -- tests if the value is in the collection, returning True/False.

  list = ['larry', 'curly', 'moe']  
  if 'curly' in list:  
    print 'yay'

The for/in constructs are very commonly used in Python code and work on data types other than list, so you should just memorize their syntax. You may have habits from other languages where you start manually iterating over a collection, where in Python you should just use for/in.

You can also use for/in to work on a string. The string acts like a list of its chars, so for ch in s: print ch prints all the chars in a string.

Range

The range(n) function yields the numbers 0, 1, ... n-1, and range(a, b) returns a, a+1, ... b-1 -- up to but not including the last number. The combination of the for-loop and the range() function allow you to build a traditional numeric for loop:

  ## print the numbers from 0 through 99  
  for i in range(100):  
    print i

There is a variant xrange() which avoids the cost of building the whole list for performance sensitive cases (in Python 3000, range() will have the good performance behavior and you can forget about xrange()).

## Accessing Values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5, 6, 7 ];

print "list1[0]: ", list1[0]

print "list2[1:5]: ", list2[1:5]

When the above code is executed, it produces the following result −

list1[0]: physics

list2[1:5]: [2, 3, 4, 5]

## Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method. For example −

#!/usr/bin/python

list = ['physics', 'chemistry', 1997, 2000];

print "Value available at index 2 : "

print list[2]

list[2] = 2001;

print "New value available at index 2 : "

print list[2]

**Note:** append() method is discussed in subsequent section.

When the above code is executed, it produces the following result −

Value available at index 2 :

1997

New value available at index 2 :

2001