List as Arrays

Presented By
M.Malarmathi
AP/IT

Lists

- A list is a sequential collection of values, it is a data structure
- Each value has a location (an index)
- Indexes range from 0 to n-1 (where n is the length of the list) and from -1 to -n
- Lists are heterogeneous = values can be of any type (strings are homogeneous because their elements are characters)

List Syntax

- Values are enclosed in [], myList = [3, ‘a’, True]
- One list can contain another
- Empty list = []
- Any type of data can be in the list
- You usually refer to a list by elements, that is with the []. You can refer to a list by its name (as one whole thing) when passing it as an argument to a function.

List semantics

- Lists are mutable, that is, elements can be changed
- Individual elements can be changed the same way any variable can be changed, with an assignment statement
- myList = [1,9, ‘a’, 4, 7]
- m = 3
- myList[m] = 99
- myList[m+1] = 88

List Operations

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort (order) the list. A comparison function may be passed as a parameter.
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i, x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the ith element of the list and returns its value.

Disadvantage of List

- Python lists are nice, but...
- They are slow to process
- They use a lot of memory
- For tables, matrices, or volumetric data, you need lists of lists of lists... which becomes messy to program.

Arrays

- multidimensional rectangular data container all elements have the same type
- compact data layout, compatible with C/Fortran
- efficient operations
- arithmetic
- flexible indexing

Why arrays?

- Arrays are the most “natural” data structure for many types of scientific data
- Matrices
- Time series
- Images
- Functions sampled on a grid
- Tables of data

List as Array

- Arrays and lists are both used in Python to store data
- They both can be used to store any data type (real numbers, strings, etc), and they both can be indexed and iterated
- **The main difference between a list and an array is the functions that you can perform to them.**

- For example, you can divide an array by 3, and each number in the array will be divided by 3 and the result will be printed.

  Example

  ```
  x = array([3, 6, 9, 12])
  x/3.0
  print(x)
  ```

  Output:

  ```
  array([1, 2, 3, 4])
  ```

- To divide a list by 3, Python will tell you that it can't be done, and an error will be thrown.

  ```
  y = [3, 6, 9, 12]
  y/3.0
  print(y)
  ```

  Output:

  ```
  Syntax error
  ```

Similarities between arrays and lists:

- Both are mutable: both can have elements reassigned in place
- Arrays and lists are indexed and sliced identically
- The `len` command works just as well on arrays as anything else
- Arrays and lists both have `sort` and `reverse` attributes

- With arrays, the + and * signs do not refer to concatenation or repetition
- Examples:

```
>>> ar1 = array([2,4,6])
>>> ar1+2 # Adding a constant to an array adds
the constant to each term
[4,6,8,]  # in the array
>>> ar1*2 # Multiplying an array by a constant
multiplies each term in
[4,8,12,] # the array by that constant
```

- Adding two arrays is just like adding two vectors

```
>>> ar1 = array([2,4,6]); ar2 = array([1,2,3])
>>> ar1+ar2
[3,6,9,]
```

- Multiplying two arrays multiplies them term by term:

```
>>> ar1*ar2
[2,8,18,]
```

- Same for division:

```
>>> ar1/ar2
[2,2,2,]
```

- Mutable types (dictionaries, lists, arrays) can have individual items reassigned in place, while immutable types (numbers, strings, tuples) cannot.

```
>>> L = [0,2,3]
>>> L[0] = 1
>>> L
[1,2,3]
>>> s = 'string'
>>> s[3] = 'o'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
```

Thank You