Lists

Darin Brezeale

The University of Texas at Arlington

Lists

While single variables have many uses, there are times in which we wish to store multiple related values. For these we may wish to use a list.

The mathematical equivalent of a one dimensional list is a vector.

Example: We may have the vector $X = (3.5, 4.0, 9.34)$ whose terms are referenced as x_1, x_2, and x_3.

Creating Lists

We create a list of objects by enclosing the objects in square brackets (i.e., [ ]). To reference a particular element of the list, we use the name of the list with the subscript of the element. Subscripts begin at 0 and increment from left to right.

Example:

```
data = [3.5, 4, 9.34]

print data[2]
```

produces

```
9.34
```

More 1D List Examples

The program

```
data = [5, 4, 3, 2]

print data

data[2] = 99

print data
```

produces

```
[5, 4, 3, 2]
[5, 4, 99, 2]
```

Iterating Through a List

We can easily iterate through a list using the `for` statement:

```
data = [5, 4, 3, 2]

for value in data :
    print value
```

produces

```
5
4
3
2
```

Accessing List Elements

We have already seen that we can access list elements using their subscripts, with the subscripts starting at 0 and incrementing from left to right.

We can also access the elements using subscripts that began with -1 and decrement from right to left:

[18,	73,	21,	52]	list
0	1	2	3	index from left
-4	-3	-2	-1	index from right

Accessing List Elements cont.

We can also *slice* a list to get a range of values:

```
oldList = [15, 33, 72, 99, 24, 61]

# slice is [start_index, end_index + 1]
newList = oldList[1:4]

print oldList
print newList
```

produces

```
[15, 33, 72, 99, 24, 61]
[33, 72, 99]
```

Accessing List Elements cont.

Leaving out one or both of the end indices allows us to get all values in a specific direction:

```
d = [23, 44, 19, 5, 61, 7]
print d[2:]

x = d[ : ]   # copy list
print x
```

produces

```
[19, 5, 61, 7]
[23, 44, 19, 5, 61, 7]
```

Useful functions

We can get the length of a list, i.e., how many elements are in it, using the `len()` function:

```
oldList = [15, 33, 72, 24]

length = len( oldList )

print "the list has", length, "elements"
```

produces

```
the list has 4 elements
```

Useful functions

The `range()` function can produce a list:

```
data = range(88, 93)
print data
print

for i in range(0, len( data ) ) :
    print data[i]
```

produces

```
[88, 89, 90, 91, 92]

88
89
90
91
92
```

List Methods

Some useful methods (think of a method as a function for a specific type of object) for lists are:

- `append(x)`
- `pop()`
- `insert(i, x)`
- `sort()`
- `reverse()`

We use the method by connecting it to the list name with a period.

See `list_methods.py` on the course website.

List Elements

Lists can consist of objects of different types: integers, strings, floating-point numbers, even other lists.

Example:

```
data = [3, 17.9, "a string", [9, 8, 7] ]

for i in data :
    print i
```

produces

```
3
17.9
a string
[9, 8, 7]
```

Lists of Lists

When accessing an element of a list within a list, we need to know which element of which list.

```
data = [ [1, 2, 3], [4, 5, 6, 7, 8] ]

print data[0][1]     # list 0, element 1
print data[1][3]     # list 1, element 3
print data[1][1:4]   # list 1, elements 1 to 3
```

produces

```
2
7
[5, 6, 7]
```

We can have lists within lists within lists...

Lists – Indexing Error

WARNING: A common error when accessing lists via subscripts is to use an index value that is too large.

```
data = [3.5, 4, 9.34]

# range(0, 4) produces the values 0 to 3
for i in range(0, 4) :
    print data[i]
```

produces

```
3.5
4
9.34
Traceback (most recent call last):
  File "ex-lists2.py", line 8, in <module>
    print data[i]
IndexError: list index out of range
```