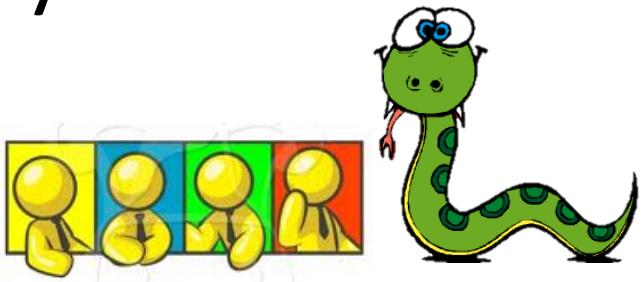
IDCE 302: Chapter 8

Lists

Jie Tian, PhD

Clark University



Outline

- Intro to list
- Create a list
- Access a list Element
- Modify a list
- for … in … structure
- Split and join

What is a list?

- A list is a **sequence of values**.
- A string can be regarded as a sequence of characters.
- The values in a list are called **elements** (or **items**).
- The elements are separated by comma (,)
- The elements in a list can be any type.

Create a List

- The simplest way to create a list is to enclose the elements in a pair of square brackets []:

```
cities= ["Boston", "Worcester"]

numbers = [17, 123]

empty = []
```

- A list **can be heterogeneous**

Nested

["hello", 2.0, 5, [10, 20]]

- Yes, empty lists [] are allowed.

Create a List (cont'd)

- Use the function **`range()`** to generate a list
- Function range() has three form (one, two, or three arguments).

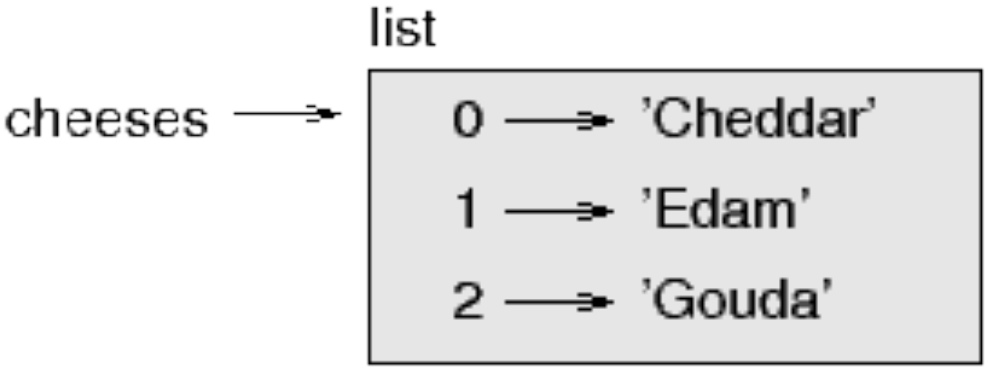
```
range(start, stop, step)
```

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(2,5)
[2, 3, 4]
>>> range(1, 10, 2)
[1, 3, 5, 7, 9]
```

Accessing Elements

- Elements in a list can be accessed using the index.

```
Cheeses=['Cheddar', 'Edam', 'Gouda']
Cheeses [0] → 'Cheddar'
Cheeses [1] → 'Edam'
Cheeses [2] → 'Gouda'
```



- If an index has a negative value, it counts backward from the end of the list:

```
>>> mylist = range(10)
>>> print mylist
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>print mylist[-1]
9
```

Nested Lists

```
>>> list = ["hello", 2.0, 5, [10, 20]]
>>> sublist= list[3]
>>> sublist[0]
10

>>> list[3][1]
20
```

Lists are mutable!

```
>>> mylist = range(10)
>>> print mylist
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> print mylist[7]
8
```

1. Change list element

```
# elements of a list can be modified
>>> mylist[7]=100

>>> print mylist
[0, 1, 2, 3, 4, 5, 6, 7, 100, 9]
```

```
>>> places = [“boston", “worcester", “natick"]
>>> places[0] = “clark"
>>> places[-1] = “wpi"
>>> print places
[“clark“, “worcester“, “wpi“]
```

2. Change list elements

```
>>> list = [“a“, “b“, “c“, “d“]
>>> list[1:3] = ['x', 'y']
>>> print list
['a', 'x', 'y', 'd']
```

3. Insert elements

```
>>> list = ['a', 'd', 'f']
>>> list[1:1] = ['b', 'c']

>>> print list
['a', 'b', 'c', 'd', 'f'] # what list[1]= ['b', 'c'] results?


>>> list[4:4] = ['e'] # Would list[4]= ['e'] work as well?

>>> print list
['a', 'b', 'c', 'd', 'e', 'f']
```

4. Delete elements

```
>>> list = ['a', 'b', 'c', 'd', 'e', 'f']
>>> list[1:3] = []

>>> print list
['a', 'd', 'e', 'f']
```

Use the del operator

- Assign the elements to be deleted with an empty list.

```
>>> a = [“one“, “two“, “three“]
>>> del a[1] # same as a[1:2]=[]
>>> a
[“one“, “three“]

>>> list = [“a“, “b“, “c“, “d“, “e“, “f“]
>>> del list[1:5]
>>> print list
[“a“, “f“]

>>>list=range(10)
>>>list
[0,1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>del list[3:]
>>>list
[0,1, 2]
```

Extending a List

- append adds **a new element** to the end of a list

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t
['a', 'b', 'c', 'd']
```

- extend takes **a list** as an argument and appends all of the elements:

```
>>> t1 = ['a', 'b', 'c']  # Would t1.append(t2) work as
>>> t2 = ['d', 'e']       well?
>>> t1.extend(t2)
>>> print t1
['a', 'b', 'c', 'd', 'e']
```

Example of Using a List

```
def printColors():
    colors = ["red", "green", "blue"]
    i = 0
    while i<3:
        print colors[i]
        i = i+1
printColors()
```

```
Output:
red
green
blue
```

Only good for the lists with 3 elements.

- **`len()` function returns the # of elements**

```
def printList(listVal):
     i = 0
     while i< len(listVal):
          print listVal[i]
          i = i+1
     print

# test cases
colors = ["red", "green", "blue"]
shapes = ["point", "line", "polygon", "prism"]
printList(colors)
printList(shapes)
```

Lists and for Loops

- the **for** … **in** … structure is very useful.
- Run a block of code for every element in a list.

```
"""
Purpose: print the items in a list using a for loop
Input:
    listVal: a list
"""
def printList(listVal):
    for listItem in listVal:
        print listItem, '\t',

colors = [["red", "green", "blue"], 'yellow', 'cyan', 'magenta']
shapes = ["point", "line", "polygon", "prism"]
printList(colors)
printList(shapes)
```

```
['red', 'green', 'blue']   yellow   cyan   magenta
point  line   polygon  prism
```

List Operations (+, *)

The '+' operation concatenate lists into one

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

The '*' operation repeats a list a given number of times

```
>>>[0] * 4
[0,0, 0, 0]
>>>[1, 2, 3] * 3
[1,2, 3, 1, 2, 3, 1, 2, 3]
```

```
# review: string related operations
>>> strName = 'road'+'.shp'
>>> strVal
'road.shp'
>>> strVal = 'apple'
>>> print strVal*3
appleappleapple
```

List Membership

- The **in** operator to check membership

```
>>> features = [“point”, “line”, “polygon”]
>>> features
['point', 'line', 'polygon']

>>> 'point' in features
True

>>> 'Point' in features # case sensitive
False
```

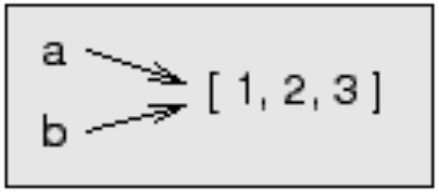
List Slicing

```
>> list = [“a“, “b“, “c“, “d“, “e“, “f“]

>>> list[1:3]
['b', 'c']

>>> list[:4]
[’a', ’b', ’c', ’d']

>>> list[3:]
['d', 'e', 'f']

>>> list[:]
['a', 'b', 'c', 'd', 'e', 'f']

# recall the string slicing operations!
```

Copying a List

- Shallow copying

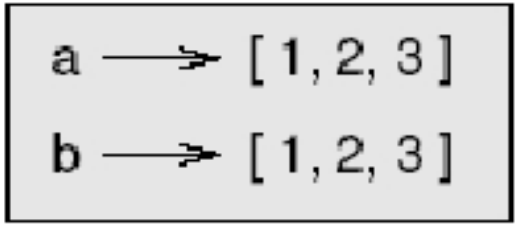
```
>>> a = [1, 2, 3]
>>> b = a
>>> a[0]= 5
>>> b
[5, 2, 3]
```



- Deep copying

Copying a List

- Shallow copying
- Deep copying (using [:])




```
>>> c = [1, 2, 3]
>>> d = c[:]
>>> c[0]= 5
>>> c
[5, 2, 3]
>>> d
[1, 2, 3]
```

Shallow Copying

```
>>> a = [1, 2, 3]
>>>  b = a
>>> b[0] = 5
>>>  print a
[5, 2, 3]  #change in b affects a
```

Deep Copying

```
>>> a = [1, 2, 3]
>>> b = a[:]  # list "b" cloned its own copy
>>> print b
[1, 2, 3]
>>> b[0] = 5

>>> print a   # list "a" is not affected
[1, 2, 3]

>>> print b
[5, 2,  3]
#  lists a and b are two  independent instances/objects
```

Example: list as argument

```
def head(list):
    return list[0]

def tail(list):
    return list[len(list)-1]

def deleteItem(list, iIndex):
    # limit the index as positive values
    if iIndex<0 or iIndex>=len(list):
        print 'Index is out of bounds'
    else
        del list[iIndex]

numbers = [1, 3, 5, 7, 9]
print head(numbers)
print tail(numbers)
deleteItem(numbers, len(numbers)/2)
print numbers
```

Output:

```
1
9
[1, 3, 7, 9]
```

```
def ParamByRef(val):
   print "\nOriginal values in list: " + str(val[0])+", "+ \
           str(val[1])
   val[0] = val[0] + 100
   val[1] = val[1] + 100
   print "After adding 100 to list elements: " + \
         str(val[0])+", "+ str(val[1])

num = [10, 20]

ParamByRef(num)

print "After function call: "+str(num[0]) + ", "+ str(num[1])
```

Output:

Original values in list: 10, 20
After adding 100 to list elements: 110, 120
After function call: 110, 120

Matrices

1	2	3
4	5	6
7	8	9

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> matrix[1]
[4, 5, 6]

>>> matrix[1][1]
5
```

Strings & Lists

- Split a string into **a list**

```
myString  = 'The autumn in New England'
mylist    = mystring.split()
print mylist
['The', 'autumn', 'in', 'New', 'England']
```

```
mylist = mystring.split('in')

print mylist
['The autumn ', ' New England']
```

- Join list elements to form a string

```
mylist  = ['The', „autumn', 'in', 'New', 'England']
mystr   = '_'.join(mylist)
print mystr
```

Connector

```
The_autumn_in_New_England
```

Summary

- Lists are very very useful!
- Lists are mutable, so you can access, change, insert and delete list elements.
- for … in … loop widely used in Python
- Shallow copying: two variables refer to a same object; deep copying: two variables refer to two different objects.
- Useful string functions (extend, join, split)
- Nested lists are used to represent matrices.

In-class exercise

Assume 5 values are given in a string as follows. Write a program to get the smallest value, the largest value, and the total value.

“2.2, 8.8, 5.5, 9.9, 1.1”

[Hint: Use the function *string.split(…)* to convert the string into a list; then traverse the list to get the numbers.]