01-4 Python Lists

CSI 500

Spring 2018

Note: course material adopted loosely from :
Downey, Allen B. *Python for software design: how to think like a computer scientist*. Cambridge University Press, 2009.
http://greenteapress.com/wp/think-python/

A list is a sequence

- A list is a sequence of elements (also called items)
  - Individual elements of the list are accessed using the bracket operator
  - the expression in the bracket is called the "index"
  - indexes start at 0, not 1 as you might expect
  - index may be an integer or an expression; will fail if non-integer
- Easy way to create a list is by enclosing items in brackets

```
# list with integers
>>> [ 10, 20, 30, 40 ]

# list with strings
>>> [ 'crunchy frog', 'ram bladder', 'lark vomit']

# list with differing types, including a list
>>> [ 'spam', 2.0, 5, [ 10, 20, 30, 40 ] ]

# lists can be assigned to variables
>>> cheeses = [ 'cheddar', 'edam', 'gouda']
>>> numbers = [ 42, 123 ]
>>> empty = []
>>> print( cheeses, numbers, empty )
['cheddar', 'edam', 'gouda'] [42, 123] []
```

Lists are mutable

- unlike strings, in Python lists are mutable
  - the index operator [] is used to access list elements
  - any integer expression can be used to index a list
- the in operator also works for lists
  - returns True or False indicating list membership

```
>>> numbers = [ 42, 123 ]
>>> numbers
[ 42, 123 ]

# let's update the first element
>>> numbers[ 1 ] = 5
>>> numbers
[ 42, 5 ]

# let's test for membership
>>> cheeses = [ 'Cheddar', 'Edam', 'Gouda' ]
>>> cheeses
[ 'Cheddar', 'Edam', 'Gouda' ]
>>>'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

Traversing a list

- A for loop can be used
  - does not require an explicit loop control variable
  - used with len() and range() functions when actual index is needed
- Lists can contain nested lists
  - each item only counts as one element

```
>>> cheeses = [ 'Cheddar', 'Edam', 'Gouda' ]
>>> for ch in cheeses:
            print ( 'I like ', ch )
I like  Cheddar
I like  Edam
I like  Gouda

>>> numbers = [1, 3, 5]
>>> for i in range( len(numbers) ):
            print('index = ', i, 'val = ', numbers[i] * 2)

index = 0, val =   2
index = 1, val =   6
index = 2, val = 10

# complex list containing lists
>>> example = ['spam', 1, ['Brie', 'Cheddar'], [1,2,3] ]
>>> len( example )
4
```

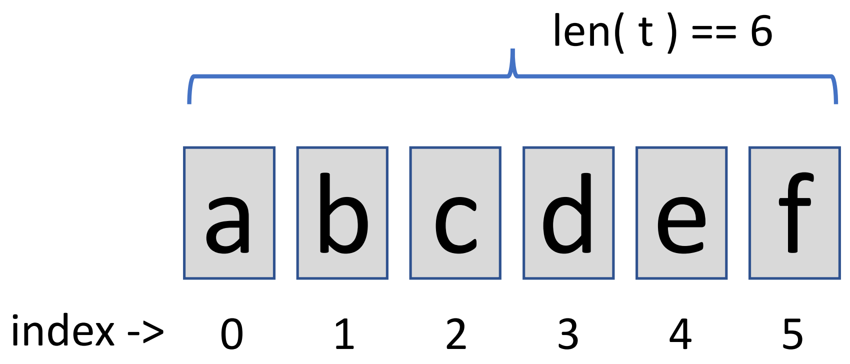
List operations

- The + operator combines lists
- The * operator repeats lists

```
# combine two lists
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]

# repeat a list
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

List slices

- A slice is a segment of a list
- The [n:m] operator is used to create slices
    - n is the first index
    - m is the last index (slices to m-1)
- [:m] starts at index 0, goes to m-1
- [n:] starts at n and goes to len( ) - 1



```
# slice from the middle
>>> t = ['a', 'b', 'c', 'd', 'e', 'f' ]
>>> t[ 1:3 ]
['b', 'c' ]

# slice from the beginning up to a value
>>> t[ :4 ]
['a', 'b', 'c', 'd']

# slice from value to the end
>>> t[ 3: ]
['d', 'e', 'f']

>>> # what will this do?
>>> v = t[ : ]
>>> v

# a slice operator on the left allows multiple assignment
>>> t[ 1:3 ] = ['x, 'y' ]
>>> t
['a', 'x', 'y', 'd', 'e', 'f']
```

List methods

- Lists are also Python "objects"
  - they have associated methods, e.g.
  - append()
    - add a single element to the end of a list
  - extend()
    - add a list to the end of a list
  - sort()
    - arrange the elements in order
- Note: list methods work "in-place"
  - they don't return a meaningful value
  - common error: overwrite your list with NoneType from a list operation!

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

>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e' ]
>>> t1.extend( t2 )
>>> t1
['a', 'b', 'c', 'd', 'e']
>>> t2                  # t2 is unchanged!
['d', 'e']

>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()            # sort works 'in-place'
>>> t
['a', 'b', 'c', 'd', 'e']

>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t = t.sort()        # rookie mistake: don't do this!
>>> t
>>> type(t)
<class 'NoneType'>
```

Map and reduce

- Python provides the += shorthand operator to increment variables
  - often used for "accumulators", or variables that count things
- operations that combine a group of things into a single value are called "reduce"
  - sum() is an example reduce
- operations that apply the same task to all elements of a list are called "map"
  - capitalize() is an example map

```
>>> def add_all( t ):
        total = 0
        for x in t:
            total += x
        return total

>>> t = [1, 3, 5]
>>> add_all( t )        # use our add_all function
9
>>> sum(t)              # use built-in sum function
9

>>> def capitalize_list( t )
        result = []
        for s in t:
            result.append( s.capitalize())
        return result

>>> capitalize_all( ['spam', 'monty'])
['Spam', 'Monty']
```

Deleting elements

- There are several ways to delete list elements
- the pop(k) operator removes ('pops') the k-th element and returns it
  - if you don't provide a value for k, it pops the last element and returns it
- if you just want to remove an indexed element, use del[k]

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop( 1 )     # return the 2nd element
>>> x
'b'
>>> t
['a', 'c']

>>> y = t.pop()        # return the 1st element
>>> y
'c'
>>> t
['a']

>>> t = [ 'a', 'b', 'c' ]
>>> del t[ 1 ]          # remove the 2nd element
>>> t
['b', 'c']
```

Deleting elements

- There are several more ways to delete list elements
- if you don't know the index, use the remove() operator
- if you want to remove several indexed elements, use the del[n:m] slice

```
>>> t = [ 'a', 'b', 'c' ]
>>> t.remove( 'b' )
>>> t
['a', 'c']

>>> t = ['a', 'b', 'c', 'd']
>>> del t[1:3]
>>> t
['a', 'd']
```

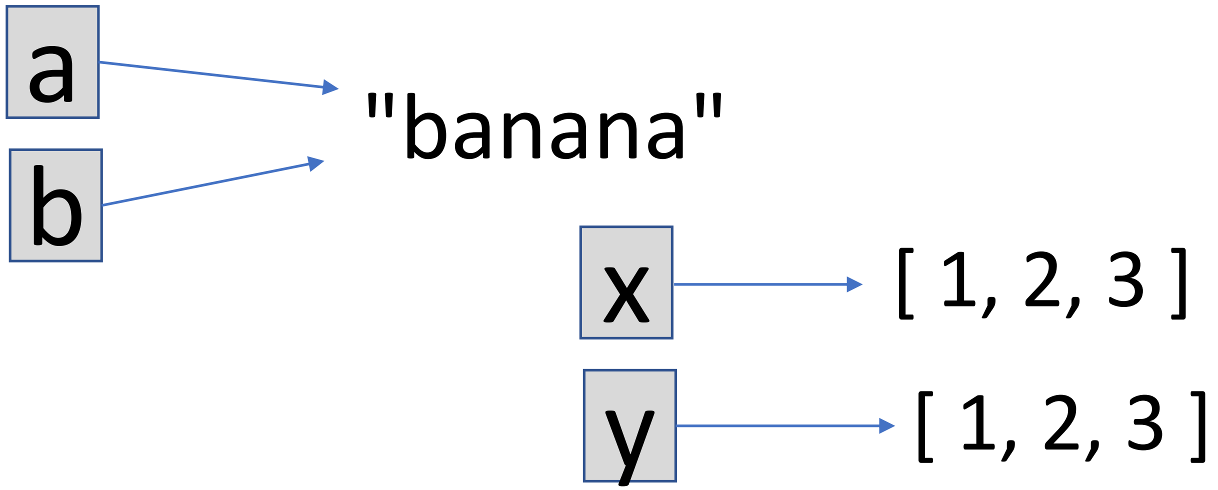
Lists and Strings

- A string is a sequence of characters
- A list is a sequence of items
- They are similar, but not the same
- the list() function coverts a string to a list
- the split() function splits a string into a list (optional delimiter)
- the join() function merges a list in to a string (optional delimter)

```
>>> s = 'spam'
>>> t = list( s )
>>> t
['s', 'p', 'a', 'm']

>>> s = 'pinin for the fjiords'
>>> t = s.split()
>>> t
['pinin', 'for', 'the', 'fjiords']

>>> s = 'spam-spam-spam'
>>> delimiter = '-'
>>> t = s.split( delimiter )
>>> t
['spam', 'spam', 'spam']

>>> t = ['pinin', 'for', 'the', 'fjiords']
>>> delimiter = '   '
>>> s = delimiter.join( t )
>>> s
'pinin for the fjiords'
```

Objects and values

- Python manages strings and lists differently
- The "is" operator tells you if two variable names refer to the same object
  - objects with more than one name are said to be "aliased"



```
>>> a = 'banana'
>>> b = 'banana'
>>> a == b
True
>>> a is b
True

>>> x = [1, 2, 3]
>>> y = [1, 2, 3]
>>> x == y
True
>>> x is y
False
```

List arguments

- If you pass a list as a function argument, the function gets a "reference" to the list
  - if you modify the list, it gets changed
- Some list actions modify lists in place
  - append()
- Other list actions create new lists
  - +, *

```
>>> def delete_head( t ):
            del t[0]

>>> t = [1,3,5]
>>> delete_head( t )              # changes t
>>> t
[3, 5]

>>> t = [1,3,5]
>>> t.append(7)                   # changes t
>>> t
[1,3,5,7]

>>> t = [1,3,5]
>>> t2 = t + [7, 9, 11]           # doesn't change t
>>> t
[1,3,5]
>>> t2
[1,3,5,7,9,11]
```

Summary

- Python lists are arrays of objects
  - Lists are mutable - they can be changed on the fly
  - Lists can contain other lists (nesting is allowed)
- List object has a variety of useful features
  - len() for length, in tests for membership
- The slice operator listvar[n:m] extracts subsets of lists
  - starts at n, goes to m-1
  - missing n assumed to be 0
  - missing m assumed to be len(listvar)-1