SAMS
Programming A/B

Week 4 Lecture – Lists
July 24, 2017

Mark Stehlik

Quiz…

- Lots of zeroes

- Indentation errors

- Returns inside loops that didn't mean to be

- Recap…
    - Run your code before submission; not at very end!
    - Print your function/variable values

- All grades should now be visible to you in autolab

Lists

- Similar to strings, but different
- String – an immutable sequence of characters
- List – a mutable sequence of data values

Representing Lists in Python

We will use a list to represent a collection of data values.

```
scores = [78, 93, 80, 68, 100, 94, 85]

colors = ['red', 'green', 'blue']

mixed = ['purple', 100, 90.5]
```
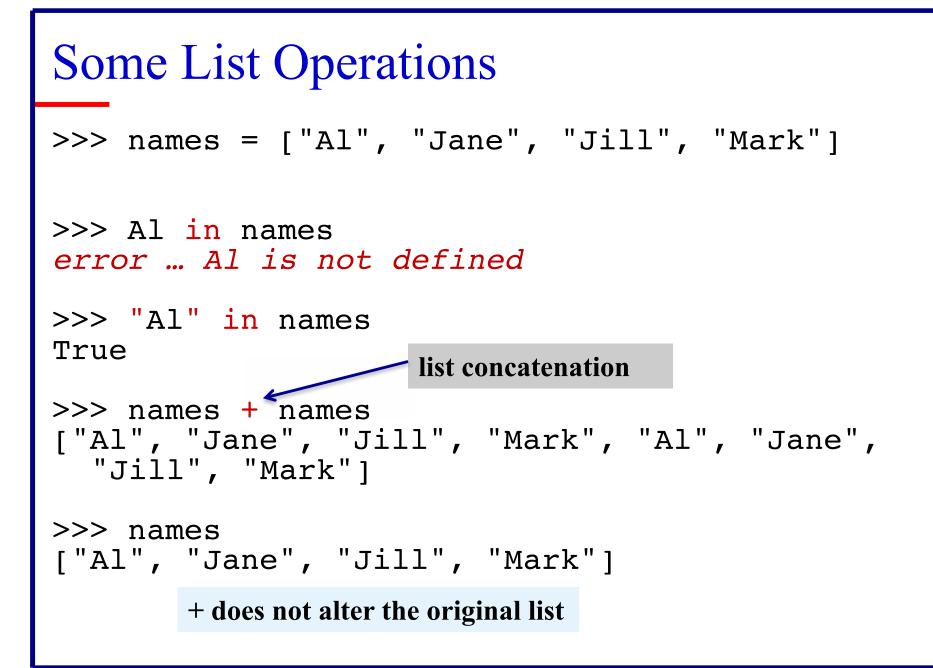
A list is an *ordered* sequence of values and may contain values of any data type.

In Python lists may be *heterogeneous* (may contain items of different data types).

Lists

- More examples:
  - Empty list
    - a = [ ]
    - a = list()
  - A way to create multiple duplicate elements
    - a2 = [" "] * 9 # how is this different from s = " " * 9??
    - arr = [0] * 5

Some List Operations

```
>>> names = ["Al", "Jane", "Jill", "Mark"]

>>> Al in names
error … Al is not defined

>>> "Al" in names
True

>>> names + names
["Al", "Jane", "Jill", "Mark", "Al", "Jane",
  "Jill", "Mark"]

>>> names
["Al", "Jane", "Jill", "Mark"]
```

list concatenation

+ does not alter the original list

Some List Operations (continued)

```
>>> a = ["A", "B", "C"]

>>> a += a

a -> ["A", "B", "C", "A", "B", "C"]
```

Accessing the elements of a list

- Indexing

  a = [2, 4, 6, 8, 10, 12]

  print(a[0], a[3], a[6]) # a[6[is an index error

  print(a[-1], a[-2])

- Valid indexes (as with strings) are

  -len .. 0 .. len-1

- Slicing, too

  a[1:3] -> [4, 6]

  a[2:] -> [6, 8, 10, 12]

List Functions

- Like strings, lists have a length
  - print(len(a))

- But also other functions
  - max, min, list, sum
  - arr = list(range(10)) produces
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  - list("Mark") produces what?

List membership

- How to test to see if something is in the list

  ```
  def search(target):
      for i in range(len(list)):
          if (list[i] == target):
              return True
      return False
  ```

- Or use "in/not in"

  ```
  def search(target):
      return target in list
  ```

List membership

- Another way to loop over a list:

```
def search(target):
    for value in list:   # no index here, just each value from first to last
        if (value == target):
            return True
    return False
```

List functions (and two methods)...

Operation	Result
x in s	True if an item of s is equal to x, else False
x not in s	False if an item of s is equal to x, else True
s + t	the concatenation of s and t
s * n, n * s	n shallow copies of s concatenated
s[i]	ith item of s, origin 0
s[i:j]	slice of s from i to j
s[i:j:k]	slice of s from i to j with step k
len(s)	length of s
min(s)	smallest item of s
max(s)	largest item of s
s.index(i)	index of the first occurence of i in s
s.count(i)	total number of occurences of i in s

Lists are mutable!

- Unlike strings, you can alter the contents of a list

  a = [2, 4, 6, 8, 10, 12]

  a[0] = 1

  a[3] = 17

- You can even alter segments of the list (slices)

  a[1:3] = [3, 5, 7, 9] -> [2, 3, 5, 7, 9, 8, 10, 12]

  # Note: the new segment doesn't have to be the same length!

Lists are mutable!

- Another example (what does this do?)

   ```
   for i in range (len(a)):
       a[i] = i
   print(a)
   ```

- Yet another example: replace the elements of a with the first n odd numbers, e.g.

   ```
   a = [2, 4, 6, 8, 10, 12]
   ```

   and I want a to be [1, 3, 5, 7, 9, 11]

   write the code to change the values of a…

   (in above loop, replace a[i] = i with a[i] = a[i] – 1)

Lists aliases…

- ## Create a list

  a = [1, 2, 3]

- ## Assign it to another variable

  b = a

- ## The second variable references the same list (b said to be an *alias* for a)

  print(b)

  b[2] = 17

  print(a, b)

Lists aliases and functions…

- You're not going to like this, but function parameters are aliases as well (unlike simple types)

```
def double(a):
    for i in range(len(a)):
        a[i] = 2 * a[i]
a = [1, 2, 3]
double(a)
print(a)
```

- So changes to a list parameter are seen outside the function

List methods (some alter the list)...

Operation	Result
`s[i] = x`	item *i* of *s* is replaced by *x*
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
`del s[i:j]`	same as `s[i:j] = []`
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list
`s.append(x)`	same as `s[len(s):len(s)] = [x]`
`s.extend(x)`	same as `s[len(s):len(s)] = x`
`s.count(x)`	return number of *i*'s for which `s[i] == x`
`s.index(x[, i[, j]])`	return smallest *k* such that `s[k] == x` and `i <= k < j`
`s.insert(i, x)`	same as `s[i:i] = [x]`
`s.pop([i])`	same as `x = s[i]; del s[i]; return x`
`s.remove(x)`	same as `del s[s.index(x)]`
`s.reverse()`	reverses the items of *s* in place
`s.sort([key[, reverse]])`	sort the items of *s* in place