Lists in Python

CS 8: Introduction to Computer Science, Winter 2018
Lecture #10

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- Homework #5 is due today
- Homework #6 is out and **DUE on MONDAY (3/5)**
- Lab #4 for tomorrow – due on Wed.
- Project #2 will be issued by the end of the week

Starting chapter 4

Sequential Data Types

- Data types that are made up of other data types
- *Example*:

  Strings are made up of character elements

- Strings are **immutable**
  - You can't exchange a character in strings by simple assignment
  - *Example*:

    Let's say, **s = 'book'**, you cannot issue **s[3] = 'm'** and expect the string **s = 'boom'**

    (it won't work that way, you'd have to do other manipulation)

Lists – More Versatile Sequences

- **Lists** are another sequential data type

- But **unlike strings**, lists …
  - can hold **<u>any</u>** type of data (not just characters)
  - *are* mutable – legal to change **list elements**

Lists – More Versatile Sequences

- Use square brackets, **[ ]** to define a list

```
fruit = ['apple', 'pear', 'orange', 'lemon']
```

- And use **[ ]** to access elements too

  `fruit[2]` gives you `'orange'`

  – Indexing works the same as strings
    - i.e. start with [0]
  – Index ***slicing*** works the same as with strings too
    - E.g. fruit[1:] = [‘pear’, ‘orange’, ‘lemon’]
    - E.g. fruit[:1] = [‘apple’, ‘pear’]

List Examples

DEMO! Let's try it!

```
>>> li = ['abcd', 2, 3, 'efg', True, 7]
>>> li
['abcd', 2, 3, 'efg', True, 7]

>>> li[0]
'abcd'

>>> li[1] - li[2]
-1

>>> li[1] + li[0]
TypeError: cannot concatenate 'str' and 'int' objects

>>> for i in li:
        print(i)
```

Note: mixed data types can be placed inside 1 list

What will this do?

Other Operations Involving Lists

- Built-in functions like **`len`** (same as strings)
  - Use **`max`** and **`min`** for extremes (work for strings too)
  - And **`sum`** (only if all elements are number types)

- Test membership in lists, just like you can with other vars:

  `in` **`not in`**

- Some examples to try:

```
li = [5, 6, 9, -22, 0, 42]
len(li)
max(li)          9 in li
min(li)          99 in li
sum(li)          0 not in li
```

More Operations Involving Lists

- But unlike strings, can use built-in **`del`** operator:

```
fruit >>> ['apple', 'pear', 'orange']
del fruit[1]
fruit >>> ['apple', 'orange']
```

- Also can use **[ ]** with = to change elements too
  (Note: you CANNOT do that with strings…)

```
fruit[0] = 'tangerine'
fruit >>> ['tangerine', 'orange']
```

List Operations: + and *

- **+** concatentates (but both operands must be lists)

```
nums = [20, -92, 4]
nums + 9 >>> TypeError
nums + [9] >>> [20, -92, 4, 9]
```

- ***** repeats (one operand is a list, other is an int)

```
nums * [2] >>> TypeError
nums * 2  >>> [20, -92, 4, 20, -92, 4]
```

- Note: can make a **<u>list of lists</u>**, but still just 1 `nums`

```
[nums] * 2  >>> [[20, -92, 4], [20, -92, 4]]
```

  – Explained next slide

Actually, Lists Hold References

- Look at prior example a different way to see this

```
[nums, nums] == [nums] * 2 >>> True
```

- Now give a name for the list of list references

```
numList = [nums, nums]
numList >>> [[20, -92, 4], [20, -92, 4]]
```

Actually, Lists Hold References

- Delete an item from original list – see result!

```
del(nums[0])
numList >>> [[-92, 4], [-92, 4]]
```

- WHY ARE ALL OF THEM AFFECTED?!?!?!

- Look at p. 124 in textbook (especially Fig. 4.4)

Another Way To Create A List
Use: `list()`

- With no arguments, creates an empty list

```
list() >>> []
```

- Or pass any sequence as an argument

```
list(range(3)) >>> [0, 1, 2]
list('cat') >>> ['c', 'a', 't']
```

- Makes a copy of another list

```
nums = [-92, 4]
numsCopy = list(nums)
nums[0] = 7
nums >>> [7, 4]
numsCopy >>> [-92, 4]
```

Let's try it!

Other Built-In List Functions

DEMO!
Let's try it!

See table 4.2 in textbook: all used as ***listname.<u>function</u>( )***

- append
- insert
- pop
- sort
- reverse
- index
- count
- remove

Methods To Add/Remove List Items

- `alist.append(item)` – similar but not same as `alist = alist + [item]` – append does not make a new list, just adds an item to old list

- `alist.insert(i,item)` – inserts item at i^{th} index;
  later items' indices all move up (i.e. increased) by one (toward end)

- `alist.remove(item)` – removes first occurrence of item;
  later items' indices all move down (i.e. reduced) by one
  - You get a `ValueError` if item not in the list

- `alist.pop()` – removes *and returns* the *last* item in a list
  - `alist.pop(i)` – removes and returns **`i`**th (index) item
  - `IndexError` if empty list or **`i`** not valid for the list

Let's try it!

Some Other List Methods

- `alist.index(item)` – returns index of first occurrence of item
  - `ValueError` if item not in the list
- `alist.count(item)` – returns number of occurrences of item in the list
- `alist.sort()` – sorts list items by value into ascending order (gives you an error if items not comparable)
  IT ALSO CHANGES `alist`!
- `alist.reverse()` – reverses the order of all items in the list
  IT ALSO CHANGES `alist`!
- **Q. How can we sort items into descending order?**

Making a List by `split`ting a String

- A handy string method named **`split`** returns **a list of substrings**
  - Example: **`string = "once upon a time"`**,
    so **`string.split() = ['once', 'upon', 'a', 'time']`**

- Application for split: count how many words are in a sentence!

```
def countWords(string):
    substrings = string.split()
    return len(substrings)
```

Modifying a `split`

- Default delimiter is white spaces
  - That is, consecutive spaces, tabs, and/or newline characters

- You CAN specify different delimiters
  - Example 1: `string = 'dog/cat/wolf/ /panther'`
    `string.split('/') = ['dog', 'cat', 'wolf', ' ', 'panther']`
  - Example 2: `string = 'Salt-N-Peppa, Rihanna, Missy Elliot'`
    `string.split(',') = ['Salt-N-Peppa', 'Rihanna', 'Missy Elliot']`

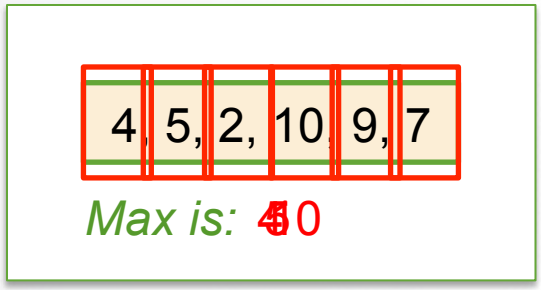
Finding Extreme Values

- Usually able to use built-in functions `max`, `min`
  - **But what if we didn't have such functions?**
  - **Or what if they don't fit our problem (e.g. max behaved oddly)?**

- Basic algorithm applies to any extreme (i.e. min OR max) finding

Use the value of first list item and call it the "extreme"
Loop through remaining items in the list:
If "current" more extreme than stored "extreme" item:
Replace stored "extreme" item with "current" value

  - Assumes there is at least one item in the list



Find-the-Maximum Algorithm

1. Store value of first list item
2. Loop through remaining items:
If current item > than stored item:
Replace stored extreme item

```
def getMax(alist):
    maxSoFar = alist[0]
    for item in alist:
        if item > maxSoFar:
            maxSoFar = item
return maxSoFar
```

Calculating Means and Medians

- Mean (Average) = (max – min) / sum
- Median (middle item) is more complex…
  - This isn't in any list function, so we have to develop it ourselves

Example:

1	5	2	10	8	7	7	6	3

sort it first and then find the middle value…

1	2	3	5	6	7	7	8	10

Median = 6

If there's an even number of entities, then employ an average calc…

1	2	3	5	6	7	7	8

Median = 5.5

“Find the Median” Algorithm

1. ***Sort*** *the list first*
2. *Determine the* ***length*** *of the list (why?)*
3. *Find the* ***middle*** *of the list (length/2)*
   a) *If the length is an* ***odd*** *number,*
      then there’s only 1 middle
   b) *If the length is an* ***even*** *number,*
      then identify the middle 2 and get their average

“Find the Median” Function

```
def median(alist):
      # Make a copy so we won't change "alist" itself
      copylist = alist
      copylist.sort() # guess what this does??

      if len(copylist)%2 == 0:   # if length of list is even, identify the middle 2 numbers
            rightmiddle = len(copylist)//2
            leftmiddle = rightmiddle - 1
            median = (copylist[leftmiddle] + copylist[rightmiddle])/2

      else:      # if length of list is odd, just find the middle number
            index_of_middle = len(copylist)//2
            median = copylist[index_of_middle]

      return median
```

YOUR TO-DOs

- ❑ Do **Homework6** (due **Monday 3/5**)
- ❑ Do **Lab4** tomorrow

- ❑ Walk on the beach

</LECTURE>