Lists and Tuples

L435/L555

Dept. of Linguistics, Indiana University
Fall 2016

Sequences

Sequences are data types where order is important

- Strings are sequences of characters
- Lists & tuples are sequence containers for more than one element, allowing for varying types
  - example: `employee = ['Markus', 'Dickinson', assistant prof.', 'BH851']`
  - (We'll focus on lists & return to tuples later)
- Each element in the sequence is assigned a position number, an index (starting from 0)
  - example: `employee[1]`
- Empty list: `[]`
  Empty string: `''`

(For a reading on lists, see:
http://greenteapress.com/thinkpython/html/thinkpython011.html)

Indexing

- Accessing elements in a list is called indexing
  - `greeting = 'hi there'`
  - `greeting[3]`
  - `'hi there'[3]`
- Indexing from the end: `greeting[-2]`
- Adding (concatenating) sequences:
  `long_greeting = greeting + ' how are you'`
- Adding multiple copies to a sequence via multiplying
  - `comment = 'this is ' + 3 * 'very ' + 'good'`

Slicing

- Accessing parts of segments is called slicing
  - `long_greeting[3:6]`
  - the slice starts at the first index and goes up to the second (non-inclusive)!
- Count from the end:
  `long_greeting[-5:-1]`
- Go to the end:
  `long_greeting[4:]`
- Start at the beginning:
  `long_greeting[:6]`
- Steps are given as optional third number:
  `long_greeting[1:6:2]`

Operations on sequences

`employee = ['Markus, Dickinson', 'assistant prof', 'BH851']`

- Check membership:
  - `'BH851' in employee`
- Check length:
  - `len(employee)`
- Math operations for lists:
  - `nums = [5, 102, 13, 2, 99, 154, 7]`
  - Minimum: `min(nums)`
    (What does `min(employee)` do?)
  - Maximum: `max(nums)`
  - Summation: `sum(nums)`

Embedded lists

Note that lists can contain lists nested inside them

```
>>> mylist = ['a', 'b', ['c', 'd', 'e']]
>>> len(mylist)
3
>>> mylist[1]
'b'
>>> mylist[2]
['c', 'd', 'e']
>>> mylist[2][0]
'c'
```

Caution

Initialization

Always initialize your variables! Otherwise you may end up with a random value.

Lists are Mutable

If you perform an operation on a list, it changes the list. In contrast, tuples and strings are immutable.

Copying of Lists

You cannot simply copy a list by variable assignment because this makes them the same object.

Mutability

Unlike strings, some mutable tasks are allowed for lists:

- Change elements in list:
  - `employee[2] = 'associate prof.'`
- Delete an element:
  - `del(employee[2])`
  - `remove` and `pop` are other ways to remove elements (more later)

We will see methods later that change the contents of a list (e.g., `sort`)

Copying of lists

Note that when you assign one list to another, you are assigning them to be *the same object*

```
>>> x = [1,2,3]
>>> y = x
>>> y
[1, 2, 3]
>>> y[1] = "hello"
>>> y
[1, 'hello', 3]
>>> x
[1, 'hello', 3]
```

The proper way to copy is `y = x[:]`

Queues & stacks

FIFO and LIFO

LIFO Last in, first out (stack)

FIFO First in, first out (queue)

Queue & stack operations

- Add at the end: append
  `employee.append('Computational Linguistics')`
- Retrieve from the end: pop
  `employee.pop()`
  - This returns a value!
- Add at the beginning:
  `employee.insert(0, 'Linguistics')`
- Retrieve from the beginning:
  `employee.pop(0)`

Sorting

- Sort destructively
  `nums.sort()`
  - Caution: this does not return a value but modifies the list itself!
  - wrong: `nums_sorted = nums.sort()`
- Non-destructive version:
  `nums_sorted = sorted(nums)`

More list methods

`x = ['a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose']`

- Count how often the same element is in a list: `count` `x.count('rose')`
- Find the first occurrence of an element in the list: `index` `x.index('rose')`
- Add a list destructively: `extend` `employee.extend(['a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose'])`
- `insert` revisited: insert anywhere ... `employee.insert(2, 'Linguistics')`
- `remove` a particular item: `x.remove('rose')`

Traversing a list

Basic way to traverse a list:

```
>>> pb_pairings = ['honey', 'peach jam',
                   'nutella', 'marmite', '']
>>> for topping in pb_pairings:
...     print(topping)
...
honey
peach jam
nutella
marmite
```

In the next couple of weeks, we'll look into this more

List comprehensions

Python has a cool shorthand called **list comprehensions** for creating new lists from old ones:

- `a = [1,2,3,4,5]`
  `b = [x**2 for x in a]`
  `b` is set to `[1, 4, 9, 16, 25]`

We'll discuss these more when we get to `for` loops

Tuples

Definition

Tuples are very similar to lists but are immutable. So once you create them, that's it!

- Indexing and slicing work with tuples just as with lists.
- Tuples do not support methods such as sorting.
- You can create them with parentheses: `mytuple=(10,50, 'foo' )`
  - `tuple` converts between types: `tuple([1,2,3])`, `tuple('abc')`
- Why bother?
  - Immutability allows for some things to be more efficient (more later) ...