LECTURE 3: DATA STRUCTURES

LISTS, TUPLES, SETS, DICTIONARIES

Introduction to Scientific Python, CME 193
Jan. 23, 2014
Download today's examples from:
`web.stanford.edu/~ermartin/Teaching/CME193-Winter15`

Eileen Martin

Outline

- Tuples
- Lists
- Mutability
- Sets
- Dictionaries
- Discuss 2nd assignment (on functions, unit testing)

Tuples

- Tuples are a sequence of values:

```
>>> a = 12, 'A', False  # example of packing
```

- They can be nested:

```
>>> b = a, 1
>>> b
((12, 'A', False), 1)
```

- They can be treated like variables and returned by functions
- They can be ***unpacked***:

```
>>> c, d = b
>>> c
(12, 'A', False)
```

- Run the code in coordinates.py

Outline

- Tuples
- Lists
- Mutability
- Sets
- Dictionaries
- Discuss 2nd assignment (on functions, unit testing)

Lists

- **Lists are a data structure that groups variables together**
- They can be heterogeneous (different types)

```
placeID = [‘CA’, 94305, ‘Stanford’]
```

- Access entries with square brackets [ ] like strings:

```
>>> placeID[:2]
[‘CA’, 94305]
```

- They can be enumerated

```
>>> list(enumerate(placeID))
[(0, ‘CA’), (1, 94305), (2, ‘Stanford’)]
```

Our first list, range(), and some ex’s

- In lecture 1 and assignment 1 you learned to use `range()`:

```
>>> range(10, 24, 2)
[10, 12, 14, 16, 18, 20, 22]
```

- Some examples of lists:

```
names = [‘Alice’, ‘Bob’, ‘Carla’]
decisions =  [‘maybe’, True, False, 0, 1, 2, ‘many’]
foods = [[‘orange’, ‘apple’, ‘banana’], [‘steak’, ‘chicken’]]
```

Properties of lists

- Can be *heterogeneous* (elements of different types):

```
decisions =  [‘maybe’, True, False, 0, 1, 2, ‘many’]
```

- Can contain any type of elements, including lists:

```
foods = [[‘orange’, ‘apple’, ‘banana’], [‘steak’, ‘chicken’]]
```

- Lists are *sequential* (have an ordering). Entries can be accessed with square brackets.

```
>>> decisions[0]
‘maybe’
```

- Try this:

```
>>> meats = [‘steak’, ‘chicken’]
>>> numberedMeats = list(enumerate(meats))
```

- What does numberedMeats look like?
- What is the type of numberedMeats?
- What’s the type of each element of numberedMeats?

Traversing lists

- Open up foodList.py and run this script:

- Can touch each element of list without knowing how long the list is.

```
fruits = ['apple', 'orange', 'banana']        # create list of strings
meats = ['steak', 'chicken']                  # create list of strings
foods = [fruits, meats]                       # create list of lists

print("Print each element of foods:")
for someList in foods:
        print(someList)

print("Print each element of foods, and associate an index:")
for index, someList in enumerate(foods):
        print("index: "+str(index)+", list: "+str(someList))
        print("foods[index]: "+str(foods[index]))

print("Print each element of each element of foods:")
for someList in foods:
        for someString in someList:
                print(someString)
```

Useful list methods example

- Run the code in listMethods.py, and open up the file in a text editor
- Useful common list methods:
  - index
  - insert
  - remove
  - pop
  - append
  - reverse
  - sort

Maps and filters applied to lists

- We can apply a function to all elements of a list with a ***map***

```
>>> somePrimes = [2, 11, 5, 3, 7]
>>> map(lambda x: x+1, somePrimes)
[3, 12, 6, 4, 8]     # returns a new list
>>> somePrimes
[2, 11, 5, 3, 7]     # doesn't change original
```

- We can look at subsets with a ***filter***

```
>>> filter(lambda x: x > 3, somePrimes)
[11, 5, 7]           # returns a new list
>>> somePrimes
[2, 11, 5, 3, 7]     # doesn't change original
```

List comprehensions

- We can use ***list comprehensions*** as another way to create lists from lists, and can be more concise and easier to understand than maps and filters

```
>>> somePrimes = [2, 11, 5, 3, 7]
>>> a = [x+1 for x in somePrimes]
>>> a                  # a new list
[3, 12, 6, 4, 8]
>>> somePrimes
[2, 11, 5, 3, 7]       # doesn’t change original
```

Outline

- Tuples
- Lists
- Mutability
- Sets
- Dictionaries
- Discuss 2nd assignment (on functions, unit testing)

Mutability

- Run the code in addOne.py:
- What does addOne() do to myList?
- Until now, we've worked with ***immutable*** objects that can not have their values changed. Tuples & strings are immutable.
- Lists (also sets, dictionaries) are ***mutable*** and can have their entries changed

Outline

- Tuples
- Lists
- Mutability
- Sets
- Dictionaries
- Discuss 2nd assignment (on functions, unit testing)

Sets

- Sets are an unordered (no indexing) group with no repeats:

```
>>> fruits = [‘apple’, ‘orange’, ‘banana’, ‘orange’]
>>> fruitSet = set(fruits)
>>> fruitSet
set([‘apple’, ‘banana’, ‘orange’])
```

- Although unordered, sets can be traversed like a sequence in a for loop
- Sets support many logical operations. For examples, run logic.py

Outline

- Tuples
- Lists
- Mutability
- Sets
- Dictionaries
- Discuss 2nd assignment (on functions, unit testing)

Dictionaries

- Dictionaries are unordered groups of ***key: value*** pairs:

```
times = {'Annabelle': 15.3, 'Bert': 14.0, 'Charlene': 14.2, 'Davis': 14.9}
```

- The keys should be unique
- Dictionaries are mutable
- For examples, Try out raceTimes.py

Discussing second assignment

- Your questions?
- Third assignment posted, due Friday 12:50 pm
  - Unit test requirements