# Python Applications

manipulating `listss`

CS101 Lecture #14

2016-11-9

# Administrivia

# Administrivia

- Homework #6 is out; due on Nov. 13.
- Last CodeLab homework
- Complete your profile info. on CodeLab

# Warmup question

# *Mutability Question*

Which of the following sets of `list` methods *all* change the `list` in place (have no return value)?

A `split`, `append`, `extend`

B `del`, `index`, `upper`

C `read`, `readlines`, `close`

D `sort`, `reverse`, `append`, `extend`

# *Mutability Question*

Which of the following sets of `list` methods *all* change the `list` in place (have no return value)?

A `split`, `append`, `extend`

B `del`, `index`, `upper`

C `read`, `readlines`, `close`

D `sort`, `reverse`, `append`, `extend` ⋆

# Working with Containers

# lists and dicts

| list<br>modifies in place | returns value |
|---|---|
| append | index |
| extend | count |
| reverse | |
| sort | |
| del (not method) | |

# lists and dicts

| `dict` modifies in place | returns value |
|---|---|
| `del` (not method) | `values`<br>`keys` |

# lists and dicts

| `dict` modifies in place | returns value |
|---|---|
| `del` (not method) | `values`<br>`keys` |

- Note that there isn't a `sort` method!

# Sorting a `dict` by value

```
# remember me?
def sortDictAsList( d ):
    items = list( d.items() )
    items.sort( key=lambda x:x[1] )
    return items

d = { 'a':2, 'b':1, 'c':-1, 'd':14 }
sortDictAsList( d )
```

# Accessing `lists`

- Sometimes we have two `lists` that correspond to each other.
- If we want to loop over both together, we have two approaches open:

```
qs = [ 'name', 'quest', 'favourite colour' ]
as = [ 'Lancelot', 'the Holy Grail', 'blue' ]
# method 1:
for i in range(len(qs)):
    print( 'What is your %s?  It is %s.'%(qs[i],as[i]) )
```

# Accessing `lists`

- Sometimes we have two `lists` that correspond to each other.
- If we want to loop over both together, we have two approaches open:

```
qs = [ 'name', 'quest', 'favourite colour' ]
as = [ 'Lancelot', 'the Holy Grail', 'blue' ]
# method 1:
for i in range(len(qs)):
    print( 'What is your %s?  It is %s.'%(qs[i],as[i]) )
```

```
# method 2:
for q,a in zip(qs,as):
    print( 'What is your %s?  It is %s.'%(q,a) )
```

# Accessing `lists`

- `zip` makes two `lists` *jointly iterable*.
- Consider a function which compares two `lists` of measurements and determines for each pair of measurements which is larger:

```
def pickLarger( a,b ):
    result = [ ]  # a list of largest values
    for i,j in zip(a,b):
        result.append( max( i,j ) )
    return result
```

# Accessing lists

- What if you need to know both an item and the index of the item?

```
my_list = [ 'meter', 'kilogram', 'second' ]
# one way
for i in range( len(my_list) ):
    print( '%s is the %sth item.' % (my_list[i]
```

# Accessing lists

- What if you need to know both an item and the index of the item?

```
my_list = [ 'meter', 'kilogram', 'second' ]
# one way
for i in range( len(my_list) ):
    print( '%s is the %sth item.' % (my_list[i]
# another way
for i,item in enumerate( my_list ):
    print( '%s is the %sth item.' % (item,i) )
```

# Accessing `lists`

- Both `zip` and `enumerate` are *convenience* functions!
- There are multiple approaches!

# Accessing `lists`

- *Permutations* are used in statistics to analyze all possible configurations of a group of things.
- In engineering, for instance, you see them used in experimental design.

```
# one way
for i in 'ABCD':
    for j in 'ABCD':
        if i == j:
            continue
        print( i, j )
```

# Accessing lists

- *Permutations* are used in statistics to analyze all possible configurations of a group of things.
- In engineering, for instance, you see them used in experimental design.

```
# one way
for i in 'ABCD':
    for j in 'ABCD':
        if i == j:
            continue
        print( i, j )

# another way
from itertools import permutations
for doublet in permutations( 'ABCD',2 ):
    print( doublet )
```

# Reminders

# *Reminders*

- Homework #6 is out; due on Nov. 13.