List Processing Patterns

What we did last time (lists)

In the previous lecture we created homogeneous, heterogeneous, and nested lists.

Today we will focus on common patterns for processing lists.

```python
def isElementOf1(val, aList):
    for elt in aList:
        if val == elt:
            return True
    return False
```

Only one of the following correctly determines if `val` is an element in the list `aList`. Which one and why?

1. `def isElementOf2(val, aList):
   for elt in aList:
       if val == elt:
           return True
   return False`

2. `def isElementOf3(val, aList):
   for elt in aList:
       if val == elt:
           return True
   return False`

```python
# A list of string lists
animalLists = [['duck', 'raccoon'], ['fox', 'raven', 'gosling'], [], ['turkey']]
# A heterogeneous list
stuff = [17, True, 'foo', None, [42, False, 'bar']]
empty = []
```

List Membership (review)

```
In []: 'Hermione Granger' in people
Out[]: True
In []: 'Hagrid' in people
Out[]: False
In []: 'Luna' in people
Out[]: False
In []: 'one G' in 'Hermione Granger'
Out[]: True
```

in and not in work on lists too

```python
people = ['Hermione Granger', 'Harry Potter', 'Ron Weasley', 'Luna Lovegood']
```

in simplifies isVowel (lec05) and isValidGesture (PS03):

```python
def isVowel(char):
    return char.lower() in 'aeiou'
def isValidGesture(g):
    return g in ['rock', 'paper', 'scissors']
```
**Review: sumList**  
(Using loops to accumulate values in a variable: sumSoFar)

```python
def sumList(nums):
    sumSoFar = 0
    for n in nums:
        sumSoFar += n
    return sumSoFar
```

In []: sumList([8,3,10,4,5])  
Out[]: 30

**Loops can be used to generate a new list**

Recall printHalves from Lec07a:

```python
def printHalves(n):
    '''Prints positive successive halves of n.'''
    while (n > 0):
        print(n)
        n = n/2
```

In []: printHalves(22)  
Out[]: 
22
11
5
2
1

**partialSums: generate a list of partial sums**

Use loops to build the list:
1. Start with an empty list []
2. Use a loop to append elements to this list one at a time

In []: partialSums([8,3,10,4,5])  
Out[]: [8,11,21,25,30]

**Exercise #1: prefixes**

```python
def prefixes(s):
    '''Given a string, returns a list of nonempty prefixes of the string, ordered from shortest to longest.'''
    prefixSoFar = ''
    prefixes = []
    for char in s:
        prefixSoFar += char
        prefixes.append(prefixSoFar)
    return prefixes
```

In []: prefixes('Paula')  
Out[]: ['P', 'Pa', 'Pau', 'Paul', 'Paula']

---

**List Patterns**

- **Step 2**: Use a loop to append elements to this list one at a time
- **Step 3**: Modify the `sumList` function to return a list of the partial sums calculated along the way:
Patterns that accumulate a new list

1. **MAPPING**: return a new list that results from performing an operation on each element of a given list.
   E.g. Return a list of the first names in `people`
   ```python
   people = ['Hermione Granger', 'Harry Potter', 'Ron Weasley', 'Luna Lovegood']
   [Hermione, Harry, Ron, Luna] → [H, H, R, L]
   ```

2. **FILTERING**: return a new list that results from keeping those elements of a given list that satisfy some condition.
   E.g. Return a list of names with last names ending in 'er' in `people`
   ```python
   people = ['Hermione Granger', 'Harry Potter']
   [Hermione, Harry] → [H, H]
   ```

Exercise #2: mapLumos

```python
def mapLumos(theList):
    '''Given a list of strings, returns a new list in which
    'Lumos' is added to the end of each string
    '''
    newList = []
    for s in theList:
        newList.append(s + 'Lumos')
    return newList
```

In [ ]: mapLumos (people)
Out[ ]: ['Hermione GrangerLumos', 'Harry PotterLumos', 'Ron WeasleyLumos', 'Luna LovegoodLumos']

In [ ]: mapLumos (['Eni', 'Sohie', 'Susan', 'Lyn', 'Ben'])
Out[ ]: ['EniLumos', 'SohieLumos', 'SusanLumos', 'LynLumos', 'BenLumos']

In [ ]: mapLumos ()
Out[ ]: []

Exercise #3: mapFirstWord

```python
def mapFirstWord(strings):
    '''Given a list of (possibly multiword) strings,
    returns a new list in which each element is the first
    word
    '''
    newList = []
    for s in strings:
        newList.append(s.split()[0])
    return newList
```

In [ ]: mapFirstWord (people)
Out[ ]: ['Hermione', 'Harry', 'Ron', 'Luna']

In [ ]: mapFirstWord (['hairy smelly dog', 'furry white bunny', 'orange clown fish'])
Out[ ]: ['hairy', 'furry', 'orange']

In [ ]: mapFirstWord (['Eni', 'Sohie', 'Susan', 'Lyn', 'Ben'])
Out[ ]: ['Eni', 'Sohie', 'Susan', 'Lyn', 'Ben']

An example of the Mapping pattern

We can produce a new list simply by performing an operation on every element in a given list. This is called the mapping pattern.

```python
def mapDouble(nums):
    '''Takes a list of numbers and returns a new list in
    which each element is twice the corresponding
    element in the input list
    '''
    result = []
    for n in nums:
        result.append(2*n)
    return result
```

mapDouble([8,3,10,5,4]) returns [16,6,20,10,8]
mapDouble([17,42,6]) returns [34,84,12]
mapDouble([]) returns []

...
An example of the Filtering Pattern

Another common way to produce a new list is to filter an existing list, keeping only those elements that satisfy a certain predicate. This is called the filtering pattern. [**[]** → [**[]**]]

```python
def filterEvens(nums):
    '''
    Takes a list of numbers and returns a new list of all numbers in the input list that are divisible by 2
    '''
    result = []
    for n in nums:
        if n % 2 == 0:
            result.append(n)
    return result

filterEvens([8,3,10,4,5]) returns [8,10,4]
filterEvens([8,2,10,4,6]) returns [8,2,10,4,6]
filterEvens([7,3,11,3,5]) returns []
```

Exercise #4: Filtering strings by containment

def filterElementsContaining(val, aList):
    '''
    Return a new list whose elements are all the elements of aList that contain val
    '''
    newList = []
    for elt in aList:
        if val in elt:
            newList.append(elt)
    return newList

people = ['Hermione Granger', 'Harry Potter', 'Ron Weasley', 'Luna Lovegood']
In []: filterElementsContaining('Harry', people)
Out[ ]: ['Harry Potter']
In []: filterElementsContaining('er', people)
Out[ ]: ['Hermione Granger', 'Harry Potter']
In []: filterElementsContaining('Voldemort', people)
Out[ ]: []
In []: filterElementsContaining('smelly', ['hairy smelly dog', 'furry white bunny', 'orange clown fish'])
Out[ ]: ['hairy smelly dog']

Nested Loops with Lists

```python
pets = ['bunny', 'cat', 'dog']
parts = ['two eyes', 'four legs', 'fur']

for pet in pets:
    for part in parts:
        print 'A', pet, 'has', part

A bunny has two eyes
A bunny has four legs
A bunny has fur
A cat has two eyes
A cat has four legs
A cat has fur
A dog has two eyes
A dog has four legs
A dog has fur
```

Exercise #5: Nested Loops with Lists

def printByCategory(categoryItemsPairs):
    '''
    Given a list of categories and a list of nested lists of items prints a category and all its corresponding items
    '''
    for pair in categoryItemsPairs:
        print pair[0] + ':'
        for item in pair[1]:
            print ' ' + item

foodCategories = [
    ('dairy', [['cheese', 'milk', 'yogurt']]),
    ('fruits', [['apples', 'bananas', 'grapes', 'oranges']]),
    ('veggies', [['cabbage', 'kale', 'lettuce']])
]