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.

```
primes = [2,3,5,7,11,13,17,19]  # List of primes less than 20
bools = [1<2, 1==2, 1>2]
houses = ['Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin']
people = ['Hermione Granger', 'Harry Potter', 'Ron Weasley', 'Luna Lovegood']

# A list of string lists
animalLists = [['duck', 'raccoon'], ['fox', 'raven', 'gosling'], [], ['turkey']]

# A heterogeneous list
stuff = [17, True, 'foo', None, [42, False, 'bar']]
empty = []  # An empty list
```

List Membership (review)

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

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

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

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

In []: 'e' in 'Hermione Granger'
Out[]: True

In []: 'one G' in 'Hermione Granger'
Out[]: False

in on strings

```
in and not in on works on lists too
people = ['Hermione Granger', 'Harry Potter', 'Ron Weasley', 'Luna Lovegood']
```

```
in simplifies isVowel (lec05) and isValidGesture (PS03):
def isVowel(char):
    return char.lower() in 'aeiou'

def isValidGesture(g):
    return g in ['rock', 'paper', 'scissors']
```

in on strings

```
x in s determines if x is a substring in s, not just if x is a character in s.
```

```
In []: 'e' in 'Hermione Granger' 0[]: True
In []: 'x' in 'Hermione Granger' 0[]: True
In []: 'Hermione' in \ 'Hermione Granger' 0[]: True
In []: 'one G' in \ 'Hermione Granger' 0[]: False
```
**Review: sumList**
(Using loops to accumulate values in a variable: sumSoFar)

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

**Iteration table**

<table>
<thead>
<tr>
<th>Step</th>
<th>n</th>
<th>sumSoFar</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>30</td>
</tr>
</tbody>
</table>

- **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
```

**Iteration table**

<table>
<thead>
<tr>
<th>Step</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

- **In [ ]: printHalves(22)**
  - **Out[]: ??**

**append plays a key role:**

```python
def halves(n):
    result = []
    while (n > 0):
        result.append(n)
        n = n / 2
    return result
```

- **In [ ]: halves(22)**
  - **Out[]: ??**

---

**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.'''
    result = []
    for c in s:
        result.append(result[-1] + c)
    return result
```

- **In [ ]: prefixes('Paula')**
  - **Out[]: ['P', 'Pa', 'Pau', 'Paul', 'Paula']**
Patterns that accumulate a new list

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

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', 'Harry', 'Ron', 'Luna']
   [Hermione, Harry, Ron, Luna] → [Hermione, Harry, Ron, Luna]
   ```

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 = ['Granger', 'Potter']
   [Granger, Potter] → [Granger, Potter]
   ```

**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'''
```

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'''
```

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 is 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. **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

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

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
def printByCategory(categoryItemsPairs):
    ''' Given a list of categories and a list of nested lists of items prints a category and all its corresponding items.
    '''

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

```python
def printByCategory(categoryItemsPairs):
    ''' Given a list of categories and a list of nested lists of items prints a category and all its corresponding items.
    '''

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

printByCategory(foodCategories)
```