PROBLEM 2: (List Comprehension (2 pts))

For this problems, assign the variable `result` to a list comprehension that calculates the answer. That means if we changed the list given, your code would still calculate the correct answer.

This must be written in one line and include a list comprehension.

The variable `result` should calculate the list of words from the list `lst`, that start and end with the same letter.

You can assume all words in `lst` contain at least one letter.

Using the list `lst` below, `result` would calculate the list: `['gong', 'roar', 'kick', 'a']`

```
lst = ['gong', 'raisin', 'roar', 'kick', 'a', 'banana']
```

```
result =
```
PROBLEM 3:  *(Splitting and Joining (2 pts))*

For this problem, use only split and join to set result to a Python expression. Do not use any Python methods. Do not use concatenation or splicing.

Given string `str` below with its assigned value, use only join and split to make the new phrase "swim*bloom*drum*charm"

You can use temporary variable(s) if you want to write the code in more than one line.

```
str = "swimmingbloommingdrummingcharm"
```

```
result =
```
Write the function named `numWords` that has one parameter named `alist`, which is a list of strings, where each string is one or more words separated by ":". This function returns the total number of words over all the strings in `alist`.

Here are several examples of calls to the function.

<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>numWords(['pop:up:red', 'blue:car', 'not:too:many:sites'])</code></td>
<td>9</td>
<td>3 + 2 + 4</td>
</tr>
<tr>
<td><code>numWords(['pop:up:red'])</code></td>
<td>3</td>
<td>3 words</td>
</tr>
<tr>
<td><code>numWords(['blue:car', 'not:too:many:sites'])</code></td>
<td>6</td>
<td>2 words + 4 words</td>
</tr>
</tbody>
</table>

Complete the function below.

```python
def numWords(alist):
```
PROBLEM 5: *(Add them up (4 pts))*

Write the function named `sumNums` that has two parameters, one integer parameter named `index` and one list parameter named `blist`, which is a list of lists of integers. Each inner list has one or more integers. This function returns the sum of the integers that are in the `index` position from each list. If a list does not have that index position, then no number from that list is considered for the sum.

Here are four examples of calls to the function. In the first example, the first and third lists `[1,6]` and `[3]` are too short, they do not have an index 2 position. The sum ignores the first list since it is too short, uses 2 from the second list `[7,3,2]`, ignores the third list since it is too short, and uses 3 from the fourth list `[11, 20, 3, 6]`.

<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sumNums(2, [[1,6],[7, 3, 2], [3], [11, 20, 3, 6]])</code></td>
<td>5</td>
<td>2 + 3</td>
</tr>
<tr>
<td><code>sumNums(2, [[7, 3, 2], [11, 20, 3, 40], [3, 1, 2]])</code></td>
<td>7</td>
<td>2 + 3 + 2</td>
</tr>
<tr>
<td><code>sumNums(1, [[7, 3, 2], [11, 20, 3, 40], [3, 1, 2]])</code></td>
<td>24</td>
<td>3 + 20 + 1</td>
</tr>
<tr>
<td><code>sumNums(3, [[7, 3, 2], [11, 20, 3, 40], [3, 1, 2]])</code></td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

Complete the function below.

```python
def sumNums(index, blist):
```