Concept Model:
Consider the following story:
There once was a dragon and a sorcerer's apprentice. The apprentice was tasked with determining which numbers in a list were odd (in contrast to even, not as in unusual), and he needed the dragon's help. The apprentice went down to the dungeons and asked, "Dragon, I need to know if any of the numbers in this list are odd: [3142, 5798, 6550, 8914]"

The dragon, being surly and rather dissatisfied with his dungeon accomodations replied:
"Sorry, I can only tell you if the first number of the list is odd."
The apprentice pleaded, "But I need to know if any number in the list is odd, not just the first!" Unmoved, the dragon retorted,
"Well, I'll only look at the first number, but I'll look at as many lists as you like."

1. What should the sorcerer's apprentice do to solve his problem?

And so, the sorcerer's apprentice presented the dragon with the original list.

Apprentice: [3142, 5798, 6550, 8914]
Dragon: The first number is not odd.

And then, the sorcerer's apprentice presented the dragon with a modified version of the list:

Apprentice: [3142, 5798, 6550, 8914]
Dragon: The first number is not odd.
Apprentice: \([3142, 5798, 6550, 8914]\)

Dragon: The first number is **not odd**.

Once more, the sorcerer's apprentice presented the dragon with a modified version of the list:

Apprentice: \([3142, 5798, 6550, 8914]\)

Dragon: The first number is **not odd**.

Finally, the sorcerer's apprentice presented the dragon with a modified version of the list:

Apprentice: \([3142, 5798, 6550, 8914]\)

Dragon: That's an **empty list**, it can't be odd!

Quite satisfied with the dragon's input, the apprentice smiled and remarked, "Ah, so none of the numbers are odd, thank you!" The dragon responded, "But how can you know that, I only told you if the first number was odd!"

2. How does the apprentice know that all the numbers are not odd?

The apprentice replied, "I gave you the following sub-lists of my original list, and you gave me an answer for the first item in each:"

\[
\begin{align*}
&[3142, 5798, 6550, 8914]\\
&[5798, 6550, 8914]\\
&[6550, 8914]\\
&[8914]\\
&\[
\end{align*}
\]

The dragon simply grumbled, "It looks like you've discovered recursion."

3. How does the dragon know when to stop checking values?

4. What is the process the dragon repeatedly executes?

5. How does the apprentice get the dragon to proceed to the next number?

6. Based on this story, how might you define recursion?
Critical Thinking Questions:

1. Examine the sample code below from interactive python which represents the problem & solution from the Concept Model story above:

```
>>> def printFirstOdd(lst):
    if lst:
        if lst[0] % 2 != 0:
            print('Odd!:', lst[0])
        printFirstOdd(lst[1:]),

>>> mylist = [3142, 5798, 6550, 8914]
>>> printFirstOdd(mylist)
```

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>def printFirstOdd(lst):</code></td>
</tr>
<tr>
<td>1</td>
<td><code>if lst:</code></td>
</tr>
<tr>
<td>2</td>
<td><code>if lst[0] % 2 != 0:</code></td>
</tr>
<tr>
<td>3</td>
<td><code>print('Odd!:', lst[0])</code></td>
</tr>
<tr>
<td>4</td>
<td><code>printFirstOdd(lst[1:])</code></td>
</tr>
<tr>
<td>5</td>
<td><code>mylist = [3142, 5798, 6550, 8914]</code></td>
</tr>
<tr>
<td>6</td>
<td><code>printFirstOdd(mylist)</code></td>
</tr>
</tbody>
</table>

a. What does each line of code do?

b. On which line is the stopping condition? ________________

c. On which lines are the small repeated steps? ________________

d. On which line is the journey broken down into smaller pieces? ________________

e. Which lines might be said to be the Dragon's actions? ________________

f. Which lines might be said to be the Apprentice's actions? ________________

g. When `myList = [3142, 5798, 6550, 8914]`, what is the argument passed to `printFirstOdd(..)` each time it's called on line 4?

```
On line 6: ________________  Third time (4): ________________
First time (4): ________________  Fourth time (4): ________________
Second time (4): ________________
```

h. If `myList = [0, 7, 9, 4, 2]`, how many times will `printFirstOdd(..)` be called? What will the arguments passed to the recursive call each time?

---

FYI: **Recursion** is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem. There are three steps to consider when building a recursive problem solution:

1. What is the stopping condition / base case?
2. What is the small, repeated step?
3. How do we break the journey down into a smaller piece?
2. Examine the sample code below from interactive python which is similar to the previous example:

<table>
<thead>
<tr>
<th>Interactive Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; def anyOdd(lst):</td>
</tr>
<tr>
<td>...     if lst:</td>
</tr>
<tr>
<td>...         if lst[0]%2!=0:</td>
</tr>
<tr>
<td>...             return True</td>
</tr>
<tr>
<td>...         else:</td>
</tr>
<tr>
<td>...             return False or anyOdd(lst[1:])</td>
</tr>
<tr>
<td>...     return False</td>
</tr>
<tr>
<td>&gt;&gt;&gt; anyOdd([2,4,6,8])</td>
</tr>
<tr>
<td>False</td>
</tr>
<tr>
<td>&gt;&gt;&gt; anyOdd([2,4,6,8,11])</td>
</tr>
<tr>
<td>True</td>
</tr>
<tr>
<td>&gt;&gt;&gt; anyOdd([2,4,7,8,6])</td>
</tr>
<tr>
<td>True</td>
</tr>
<tr>
<td>&gt;&gt;&gt; anyOdd([])</td>
</tr>
<tr>
<td>False</td>
</tr>
</tbody>
</table>

a. What does each line of code do?

0: def anyOdd(lst):
1: if lst:
2:     if lst[0]%2!=0:
3:         return True
4:     else:
5:         return False or anyOdd(lst[1:])
6: return False

b. On which line is the stopping condition?

c. On which lines are the small repeated steps?

d. On which line is the journey broken down into smaller pieces?

e. When the initial argument passed to anyOdd is [2, 4, 6, 8], as on line 7, what is passed to anyOdd(…) as an argument each time it's called on line 5?

On line 7: ___________________________ Third time (5):
First time (5):______________________ Fourth time (5):
Second time (5):__________________
f. When the initial argument passed to anyOdd is [2, 4, 6, 8], as on line 7, what is returned by anyOdd(..) each time it's called on line 5?

1st time (5): __________ or anyOdd(___________)
2nd time (5): __________ or __________ or anyOdd(___________)
3rd time (5): __________ or __________ or __________ or __________ or anyOdd(____)
4th time (5): __________ or __________ or __________ or __________ or anyOdd(____)
5th time (5): __________ or __________ or __________ or __________ or __________

h. Why does line 7 have 5 calls to anyOdd(..) and line 11's input only have 3 calls?

i. How many calls of anyOdd(..) does line 13 entail?

3. Examine the sample code below from interactive python:

```
0 def mystery1(anyString):
1    if not anyString:
2        return 0
3    else:
4        return 1 + mystery1(anyString[1:])
5 if __name__ == '__main__':
6    print(mystery1('tryl'))
```

a. What does each line of code do?

0 __________________________________________________________
1 __________________________________________________________
2 __________________________________________________________
b. On which line is the stopping condition? __________

c. On which lines are the small repeated steps? __________

d. On which line is the journey broken down into smaller pieces? __________

e. When the initial argument passed to mystery1 is 'try1', as on line 6, what is passed to mystery1(..) as an argument each time it's called on line 4?

<table>
<thead>
<tr>
<th>On line 6:</th>
<th>Third time (4):</th>
</tr>
</thead>
<tbody>
<tr>
<td>First time (4):</td>
<td>Fourth time (4):</td>
</tr>
<tr>
<td>Second time (4):</td>
<td></td>
</tr>
</tbody>
</table>

f. When the initial argument passed to mystery1 is 'try1', as on line 6, what is returned by mystery1(..) each time it's called on line 4?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_____ + mystery1(______________)</td>
<td>_____ + _____ + mystery1(______________)</td>
<td>_____ + _____ + _____ + mystery1(______________)</td>
<td>_____ + _____ + _____ + _____ + mystery1(______________)</td>
<td>_____ + _____ + _____ + _____ + _____</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery1('try1') return?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

h. If we add a line of code to the bottom, print(mystery1('try')), what will be returned by mystery1(..) each time it's called on line 4?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_____ + mystery1(______________)</td>
<td>_____ + _____ + mystery1(______________)</td>
<td>_____ + _____ + _____ + mystery1(______________)</td>
<td>_____ + _____ + _____ + _____</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery1('try') return?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

i. What does mystery1('try') return? __________

j. What might mystery1('hello') return? __________

k. What does the mystery1(anyString) function do?

Application Questions: Use the Python Interpreter to check your work
1. Write a recursive function, `recursiveListLength(anyList)`, that will return the length of a given list, `anyList`, using recursive means:

2. What does the following program do? (Step-through with examples for `fi, inSequence`).
   ```python
   def mystery2(fi, inSequence):
       if not inSequence:
           return False
       elif fi == inSequence[0]:
           return True
       else:
           return False or mystery2(fi, inSequence[1:;])
   ```

3. Write a recursive function, `countChar(ch, anyString)`, that will count the number of occurrences of a given character, `ch`, in a given string, `anyString`, using recursive means:

4. Write a recursive function, `getItem(index, start, anyList)`, that will return the element located at `index`, in a given list, `anyList`, using recursive means. It should start looking at the `index` provided in `start` and should return `None` if that index is not found: