1. This function has the same body as function \texttt{nested\_sum} that we saw earlier, except for its base case, which we’ve hidden.

```python
def nested_sum2(obj):
    """Return the sum of the items in \texttt{<obj>}, times 2.
    I.e., return the value of \texttt{nested\_sum(obj)} * 2.
    """
    if isinstance(obj, int):
        # HIDDEN
    else:
        s = 0
        for lst_i in obj:
            s = s + nested_sum2(lst_i)
        return s
```

Consider the function call \texttt{nested\_sum2([[2, [3, 1]], 4, [[1]], [10, 20]])}.

**Assuming that \texttt{nested\_sum2} works on lists of depth < 3**, trace each iteration of the loop and fill in the table below. You should not need to trace any recursive calls.

<table>
<thead>
<tr>
<th>Value of \texttt{lst_i}</th>
<th>Return value of \texttt{nested_sum2(lst_i)}</th>
<th>Value of \texttt{s} at the end of the iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. What does the call \texttt{nested\_sum2([[2, [3, 1]], 4, [[1]], [10, 20]])} return? Is it correct?

3. Are you convinced that \texttt{nested\_sum} works properly on any nested list of depth 4? If not, what would it take to convince you?

4. Are you convinced that \texttt{nested\_sum} works properly on any nested list of any depth? If not, what would it take to convince you?
def uniques(obj):
    """Return a flattened list of the numbers in obj, with no repeats."

    >>> uniques([13, [2, 13], 4])
    [13, 2, 4]
    >>> uniques([13, [13, 13], 13])
    [13]
    """
    answer = []
    if isinstance(obj, int):
        if obj not in answer:
            answer.append(obj)
    else:
        for lst_i in obj:
            # lst_i is a nested list
            answer.extend(uniques(lst_i))
    return answer