

Recursion Exercises

```
def sum_list(L):  
    """  
    Return the sum of all ints in L.  
  
    @param int|list[int|list[...]] L: possibly-nested list of ints, finite depth  
  
    >>> sum_list([1, [2, 3], [4, 5, [6, 7], 8]])  
    36  
    """  
  
    if isinstance(L, list):  
        return sum([sum_list(x) for x in L])  
    else:  
        return L
```

1. What helper methods does this function call?

sum(...), isinstance(...), sum_list(...)

2. So far, we haven't confirmed that the function works in any cases. Trace this call: `sum_list(27)`

→ 27

3. Complete the following trace of this call: `sum_list([4, 1, 8])`

```
sum_list([4, 1, 8]) --> sum( [ sum_list(4), sum_list(1), sum_list(8) ] )  
--> sum( [ 4, 1, 8 ] )  
--> 13
```

4. Trace this call: `sum_list([4])`

*→ sum([sum_list(4)])
→ sum([4])
→ 4*

5. Trace this call: `sum_list([])`

*→ sum([])
→ 0*

lists of depth 1; sums all integers

lists of depth 2:

6. Trace this call: `sum_list([4, [1, 2, 3], 8])`

→ `sum([sum_list(4), sum_list([1, 2, 3]), sum_list(8)])`
→ `sum([4, 6, 8])`
→ 18

✓

7. Trace this call: `sum_list([[1, 2, 3], [4, 5], 8])`

→ `sum([sum_list([1, 2, 3]), sum_list([4, 5]), sum_list(8)])`
→ `sum([6, 9, 8])` → 15

lists of depth 2: sums all integers

8. Trace this call: `sum_list([1, [2, 2], [2, [3, 3, 3], 2]])`

→ `sum([sum_list(1), sum_list([2, 2]), sum_list([2, [3, 3, 3], 2])])`
→ `sum([1, 4, 13])` → 18

lists of depth 3 sums all integers

9. Trace this call: `sum_list([1, [2, 2], [2, [3, [4, 4], 3, 3], 2]])`

→ `sum([sum_list(1), sum_list([2, 2]), sum_list([2, [3, [4, 4], 3, 3], 2])])`
→ `sum([1, 4, 21])` → 26

lists of depth 4: sums all ints

10. Are you a believer yet?

lets try depth 37...