

1. Review: nested lists and types

```
animals = ["cat", ["dog", "bird"], "fish"]  
animals_deep = ["cat", ["dog", ["bird", ["fish"]]]]
```

a. What does the code output? Fill out the table.

expression	value
animals[0]	
animals[1]	
animals[1][0]	
animals_deep[1][1][1][0]	

Solution: "cat"; ["dog", "bird"]; "dog"; "fish".

We never know how deep an item is. So when we find a list, we go inside it and **start again**. That is recursion.

b. `type(x) == list` tells you if x is a list. `type(x) == str` says if x is a string. What does the code output?

expression	True or False?
<code>type(animals[0]) == list</code>	
<code>type(animals[1]) == list</code>	
<code>type(animals[1]) == str</code>	

Solution: False, True, True.

2. Counting strings in a nested list

`count_strings(lst)` returns how many strings are anywhere inside a nested list of strings `lst`.

a. Start small. `count_strings(["dog", "bird"])` returns

Solution: 2

b. Now trace the whole list. Fill in the blanks.

```
count_strings(["cat", ["dog", ["bird"]], "fish"])  
= 1 + count_strings(["dog", ["bird"]]) + 1  
= 1 + (_____ + count_strings(_____)) + 1  
= 1 + (_____ + _____) + 1  
= _____
```

"cat", "dog", and "fish" are strings, so each one adds 1. ["dog", ["bird"]] and ["bird"] are lists, so we call `count_strings` again.

Solution: ["bird"], then 1, then 4.

c. Now write the code. Fill in the blanks.

```
def count_strings(lst):  
    total = 0                                # collects the answers  
  
    for x in lst:                            # visit every item  
  
        if _____:  
  
            total = total + _____  
  
        else:  
  
            total = total + _____  
  
    return total
```

Solution:

```
def count_strings(lst):  
    total = 0  
    for x in lst:  
        if type(x) == str:  
            total = total + 1  
        else:  
            total = total + count_strings(x)  
    return total
```

d. What is the base case here?

Solution: The base case is an empty list. If `lst` is empty, the `for` loop runs zero times, so `total` stays 0 and the function returns 0.

When the loop reaches a string, it does not make another recursive call—it simply adds 1. Only nested lists cause another call to `count_strings`.