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]`	

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`**?**
`type(animals[0]) == list`	
`type(animals[1]) == list`	
`type(animals[1]) == str`	

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 ____

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.

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
```

d. What is the base case here?