Recursion Exercises

```
def max_depth(obj):
    """
    Return 1 + the maximum depth of obj's elements if obj is a list.
    Otherwise return 0.

    @param object|list obj: list or object to return depth of
    @rtype: int

    >>> max_depth(17)
    0
    >>> max_depth([])
    1
    >>> max_depth([1, "two", 3])
    1
    >>> max_depth([1, ["two", 3], 4])
    2
    >>> max_depth([1, [2, ["three", 4], 5], 6])
    3
    """
    if not isinstance(obj, list):
        return 0
    elif obj == []:
        return 1
    else:
        return 1 + max([max_depth(x) for x in obj])
```

1. What helper methods does this function call?

2. So far, we haven't confirmed that the function works in any cases. Trace this call: `max_depth(27)`. **Important**: in **all** tracing examples do **not** trace an example you've already traced something similar for — immediately replace it with its value!

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

```
max_depth([4, 1, 8])  --> max( [  max_depth(4),  max_depth(1),  max_depth(8)  ] )
                      --> max( [
                      -->
```

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

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

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

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

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

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

10. Are you a believer yet?

```
def concat_strings(string_list):
    """
    Concatenate all the strings in possibly-nested string_list.

    @param list[str]|str string_list:
    @rtype: str

    >>> concat_strings("brown")
    'brown'
    >>> concat_strings(["now", "brown"])
    'nowbrown'
    >>> concat_strings(["how", ["now", "brown"], "cow"])
    'hownowbrowncow'
    """
    if isinstance(string_list, list):
        return "".join([concat_strings(x) for x in string_list])
    else:
        return string_list
```

1. What helper methods does this function call?

2. So far, we haven't confirmed that the function works in any cases. Trace this call: `concat_strings("brown")`. **Important**: in **all** tracing examples do **not** trace an example you've already traced something similar for — immediately replace it with its value!

3. Complete the following trace of this call: `concat_strings(["now", "brown"])`

```
concat_strings(["now", "brown"])  --> "".join([concat_strings("now"), concat_strings("brown")])
                                  --> "".join( [
                                  -->
```

4. Trace this call: `concat_strings(["how"])`

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

6. Trace this call: `concat_strings(["how", ["now", "brown"], "cow"])`

```
    --> "".join([
            cs("how"), cs["now", "brown"], cs("cow")
            ])
    --> "".join([
            "how", "".join(["now", "brown"]), "cow"
            ])
    --> "".join([
            "how", "nowbrown", "cow"
            ])
    --> "hownowbrowncow"
```

7. Trace this call: `concat_strings(["how", ["now", ["brown", "cow"]]])`

```
  --> (If) "".join([
        cs("how"), cs(["now", ["brown", "cow"]])
        ])
  --> "".join([
        "now", "".join([cs("now"), cs(["brown", "cow"])])
        ])
  --> "".join([
        "now", "".join(["now", "browncow"])
        ])
  --> "".join(["now", "nowbrowncow"])
  --> "nownowbrowcow"
```

8. Trace this call: `concat_strings(["one" ["two", ["three", ["four", "five"], "six"]]])`

```
--> "".join([
      cs("one"), cs(["two", ["three", ["four", "five"], "six"]])
      ])
--> "".join([
      "one", "".join(["two", cs(["three", ["four", "five"], "six"])])
      ])
--> "".join([
      "one", "".join(["two", "".join(["three", cs(["four", "five"]), "six"])
      ])
--> "".join([
      "one", "twothreefourfivesix"
      ])
--> "onetwothreefourfivesix"
```