# CSC148 - Mutating Nested Lists

To end off our study of nested lists, we're going to look at a more complex form of recursion on nested lists involving *mutation*. The running example we'll use for this worksheet is the following function:

```
def add_one(obj: Union[int, List]) -> None:
    """Add one to every number stored in <obj>. Do nothing if <obj> is an int.

    If <obj> is a list, *mutate* it to change the numbers stored.

    >>> lst0 = 1
    >>> add_one(lst0)
    >>> lst0
    1
    >>> lst1 = []
    >>> add_one(lst1)
    >>> lst1
    []
    >>> lst2 = [1, [2, 3], [[[5]]]]
    >>> add_one(lst2)
    >>> lst2
    [2, [3, 4], [[[6]]]]
    """
    # if isinstance(obj, int):
    #     ...
    # else:
    #     for sublist in obj:
    #         ... add_one(sublist) ...
```

1. To start, think about the *base case* for this function. Implement it in the space below.

   *Hint*: read the docstring carefully—it tells you exactly what to do.

   ```
   if isinstance(obj, int):
       pass
   ```

2. Now for the recursive step. The limitation of the standard `for sublist in obj` loop is that while it can mutate individual sub-nested-lists of `obj` that are lists, it can't modify any *integer* element of `obj` directly.

   To make sure you understand this, suppose `obj = [1, 2, 3]`, and we run the following code on it:

   ```
   else:
       # obj = [1, 2, 3]
       for sublist in obj:
           sublist += 1
   ```

   ```
   for i in range(len(obj)):
       obj[i] += 1
   ```

   What would be the result of executing this code?

   *Hint*: draw a memory model diagram on a separate sheet of paper. This is good review for the midterm as well!

3. In general, if we want to mutate elements of a list, we loop over the *indexes* of the list rather than its elements directly:

```
for i in range(len(obj)):
```

Using your answer to Question 1, and this new loop form here, implement `add_one` in the space below. This is still challenging, so please don't hesitate to talk to your neighbours and TAs for help!

*Hint*: you'll need different cases in your loop for when `obj[i]` is an integer or a list.

```
def add_one(obj: Union[int, List]) -> None:
"""Add one to every number stored in <obj>. Do nothing if <obj> is an int."""
if isinstance(obj, int):
    # Write your answer to Question 1 here.
    pass


else:


    for i in range(len(obj)):
            add_one(obj)
```