changmiao Yu 2024-03-03.

# 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: 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) ...
```

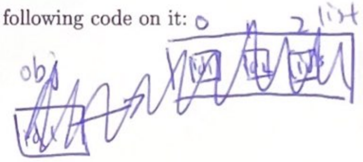
1. To start, think about the *base case* for this function. Implement it in the space below.
*Hint*: read the docstring carefully—it tells you exactly what to do.

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

we wrote the code in Pycharm.
Do nothing.

2. Now for the recursive step. The standard "`for sublist in obj`" loop can mutate individual sub-nested-lists of `obj` that are lists, it can't mutate 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
```

What is the problem with this code? (*Hint*: review our memory model diagrams!)

The list does not change. we got values from list, but did not change list.

we reminded ourselves of this for-loop issue using an exaple that was unrelated to recursion. (This was done in the Python shell)

3. In general, if we want to replace 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, implement add_one in the space below.

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

```
def add_one(obj: int | list) -> None:
    """Add one to every number stored in <obj>. Do nothing if <obj> is an int."""
    if isinstance(obj, int):    when obj is int
        # Write your answer to Question 1 here.
        # do nothing
        return None

    else:
        ~~res = [ ]~~

        for i in range(len(obj)):
            ~~add-one(obj[i])~~
            if isinstance(obj[i] int):
                obj[i] += 1

            else:
            # 还是 list
                add_one(obj[i])
```

we must notice int
and increment ourselves.

example

[1, [2, 3], [[4], 5], [[6]]]

add-one(1) → 2

add-one([2, 3]) → [3, 4]

add-one[[4], 5] → ~~[4, 6]~~ [[5], 6]

add-one[[6]] → ~~[, 8]~~ [[7]]

加起来

~~Recur~~Recursive Design recipe.

1 identify recursive structure.

2. Base Case.

(a) identity & write doc test.

(b) Implement.

3. Recursive Case.

a) write concrete example of greater complexity

b) write down recursive calls that need to be made

c) using only doc string, write down ~~those call will~~ what those calls will return

d). figure out how to combine recursive calls.

e) Implement the recursive cases

Use partial tracing when debugg. recursion