CSC148 - Writing Recursive Functions

To review, here is the basic recursive design recipe, which you got lots of practice on in this week's prep:

```python
def f(obj: Union[int, List]) -> ...:
    if isinstance(obj, int):
        ...
    else:
        ...
        for sublist in obj:
            ... f(sublist) ...
```

handy template ... but don't get too attached :)

On this worksheet, we'll look at two variation of this pattern, both based on what happens when the function we're implementing takes additional parameters beyond just a single nested list.

Here are the steps you should take, adapted from the *Design Recipe for recursive functions* from the readings:

1. Write at least one doctest for a base case (when `obj` is a single integer), and then implement the base case directly.

2. Write a doctest for the function call on an input of some complexity (e.g., a nested list of depth 3).

   At the side of the page (or on scrap paper), write down the relevant recursive calls (for each sub-nested-list of your input), and what each call would return based on the function's docstring.

   Note: think carefully about the extra parameter for each function; should it stay the same between recursive calls, or should it change somehow?

3. Use your work in the previous step to implement the recursive case in your function body.

```python
1. def nested_list_contains(obj: Union[int, List], item: int) -> bool:
       """Return whether the given item appears in <obj>.

       Note that if <obj> is an integer, this function checks whether
       <item> is equal to <obj>.

       SPACE FOR DOCTESTS BELOW
```

very simple nested list

```
>>> nested_list_contains( ??  ,  ?? )
True/False
```

more complex

```
>>> nested_list_contains( ??  ,  ?? )
True/False
```

item we are looking for in the nested list.

```python
       """
       # Your code goes here!
```

2. Recall from the readings that we defined the *depth* of a nested list as the maximum number of times a list is nested within another one in the nested list. Analogously, we can define the *depth of an object in a nested list* to be the number of nested lists enclosing the object. For example, in the list `[10, [[20]], [30, 40]]`, the depth of `10` is 1, the depth of `20` is 3, and the depths of `30` and `40` are 2. On the other hand, for a nested list that is a single integer, e.g. `100`, the depth of `100` in this nested list is 0.

Implement the function below by following the same steps as before. However, keep in mind here that when we recurse, the `d` that we pass in needs to change—how?

```python
def first_at_depth(obj: Union[int, List], d: int) -> Optional[int]:
    """Return the first (leftmost) item in <obj> at depth <d>.

    Return None if there is no item at depth <d>.

    Precondition: d >= 0.

    SPACE FOR DOCTESTS BELOW



    """
    # Your code goes here!
```

[1, [22], [2, [3,4]], 98], 3)

nested-list-contains (1, 3) → F

"   "   "   ([22], 3) → F

"   "   "   ([2, [3,4]], 3) → (T) can stop here

"   "   "   (98, 3) → F