Trace the following code and draw out the state of memory at the steps indicated below.

```python
def foo(x):
    for i in range(len(x)):
        x[i] += 1

a = [ 9, 99, 999 ]
b = a[:]
c = foo(a)
```

Remember to show all variables in the local frame for function `foo()` and in the global frame (i.e. the frame for the main program). Be sure to identify which frame is which. Your model should clearly show the values of each variable – it does not necessarily need to show which values are in the same location, but make sure you can keep track of which lists are aliases of one another.

1. State of memory just after the `foo()` function call begins executing

[Diagram showing memory state]
2. State of memory just before the foo() function call returns
3. State of memory just **after** the code finishes executing
NOTE ON SOLUTIONS:
When drawing solutions on an exam, you may find the above solutions from showing the python visualizer approach easiest to replicate—it has the benefit of clear showing which lists are copies and which are aliases. But it can get a little complicated.

So you could also draw a correct answer using the simplified memory model to represent the values in each variable, as in the alternate solution below. Just make sure you can keep track of what is an alias and what is a copy.

1. Draw the state of memory just after the `foo()` function call begins executing (Exclude functions from frame).

   Global Frame:
   ```
   a → [9, 99, 999]
   b → [9, 99, 999]
   ```

   foo frame:
   ```
   x → [9,99,999] (*alias of in global frame)
   ```

2. Draw the state of memory just before the `foo()` function call returns (Exclude functions from frame).

   Global Frame:
   ```
   a → [10, 100, 1000]
   b → [9, 99, 999]
   ```

   foo frame:
   ```
   x → [10, 100, 1000]
   i → 2
   ```

3. Draw the state of memory just after the code finishes executing (Exclude functions from frame).

   Global Frame:
   ```
   a → [10, 100, 1000]
   b → [9, 99, 999]
   c → None
   ```