Solution 15: Move Zeros Using Two Pointers

1) What does “in-place” mean?

- In-place means we **do** change the input list itself.
- We **don't** create a new list.

2) Two pointers: `read` and `write`

One correct set of answers:

- `read` starts at **0** and moves to the **right** end.
- `write` starts at **0** and points to the next place to write a **nonzero** number.
- If `numbers[read]` is zero, we only move **read**.
- If `numbers[read]` is nonzero, we copy it to `numbers[write]`.

3) Output (scan only)

Let’s track the scan step:

Start: `numbers = [0, 1, 0, 3, 0, 2]`

After scan (before filling zeros): `numbers = [1, 3, 2, 3, 0, 2]` and `write = 3`

So the output is:

```
[1, 3, 2, 3, 0, 2]
write = 3
```

4) After scan, fill zeros

We fill indices `3, 4, 5` with zeros:

Answer:

`numbers = [1, 3, 2, 0, 0, 0]`

5) Output (full algorithm)

Input: `[1, 0, 2, 0, 0, 3]`

Nonzeros in order are `1, 2, 3`, then zeros.

Output:

```
[1, 2, 3, 0, 0, 0]
```

6) Debugging (2 bugs)

Bug 1: - `while read <= n` should be `while read < n` (otherwise `read == n` is out of range).

Bug 2: - `write = write + 1` must happen **only when we copy a nonzero**.

Corrected lines:

- Line A: `while read < n:`
- Line B: `write = write + 1`

One correct fixed version:

```
numbers = [0, 1, 0, 3, 0, 2]

read = 0
write = 0
n = len(numbers)

while read < n:
    if numbers[read] != 0:
        numbers[write] = numbers[read]
        write = write + 1
    read = read + 1
```

7) Fill in the missing code (all while-loops)

One correct completion:

```
numbers = [0, 1, 0, 3, 0, 2]

read = 0
write = 0
n = len(numbers)

# Step 1: move all nonzeros to the front (keep order)
while read < n:
    if numbers[read] != 0:
        numbers[write] = numbers[read]
        write = write + 1
    read = read + 1

# Step 2: fill zeros from write to the end
k = write
while k < n:
    numbers[k] = 0
    k = k + 1

print(numbers)
```

Output:

```
[1, 3, 2, 0, 0, 0]
```