Problem Statement:

You are developing a performance-critical data movement routine for a simulated execution environment that models strict instruction timing.

The goal is to move a fixed number of elements from an input buffer into a newly allocated output region. Execution Constraints

1. Reduced Control Checks Evaluating a loop condition for every element transfer introduces unacceptable overhead. Instead, the logic must be structured so that control checks occur only after multiple transfers have been completed.

2. Fixed-Width Transfer Groups Transfers must be organized into groups of exactly eight elements. Once the system enters the repeating portion of the routine, each iteration must perform eight explicit assignments without additional branching inside the group.

3. Partial Group Handling The total number of elements to transfer may not align with the group size. Any elements that do not fit evenly into a full group must be processed before the repeating portion begins.

4. Explicit Control Flow for Partial Transfers The partial group must be handled without using a compact counting loop (e.g., no short for or while constructs). Instead, control flow must be arranged so that execution begins at a position corresponding to the required number of initial assignments and then proceeds sequentially.

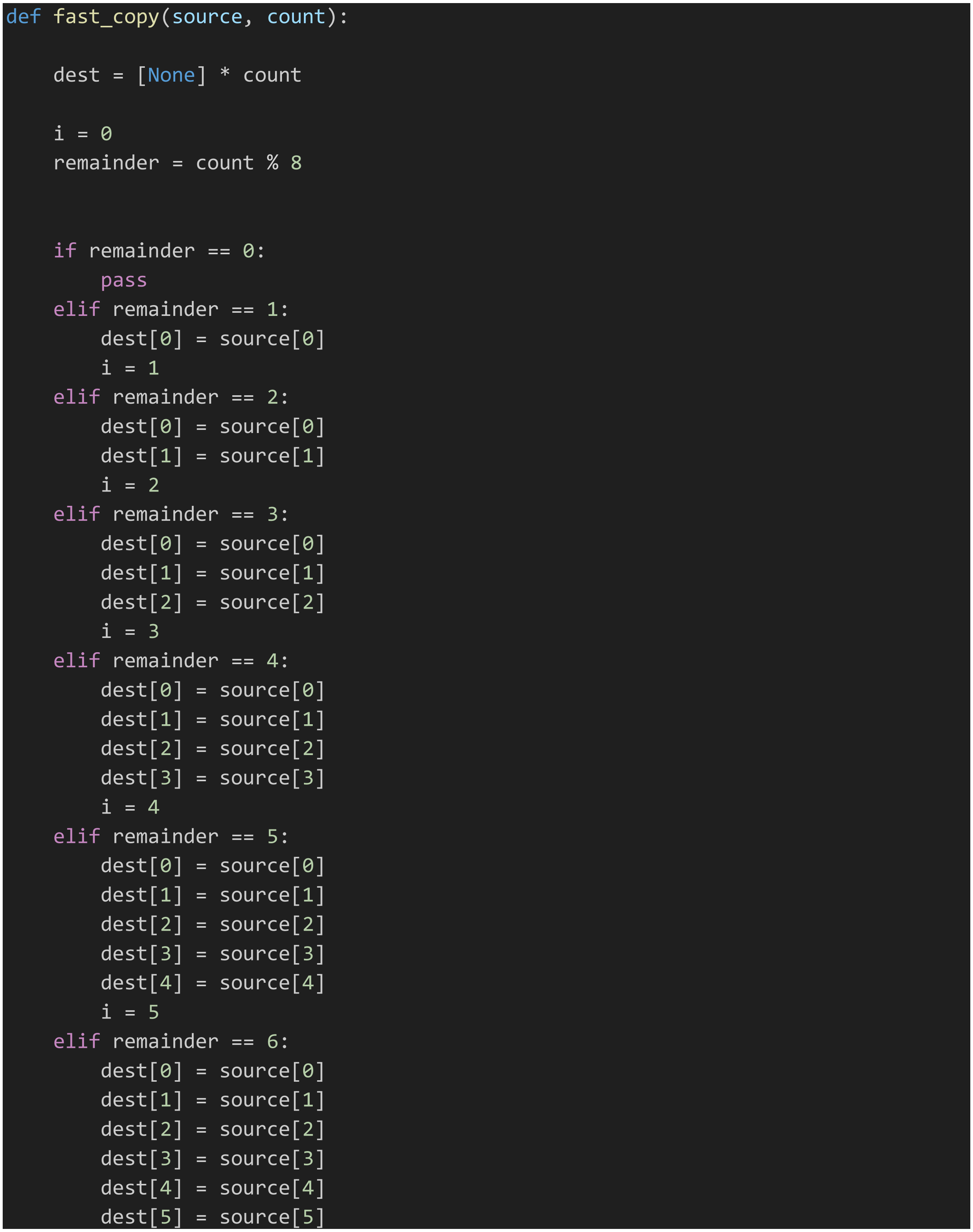
5. Element-by-Element Assignment Only High-level bulk operations are not permitted.

Each element must be copied using a single explicit assignment, simulating a register-to-memory transfer.

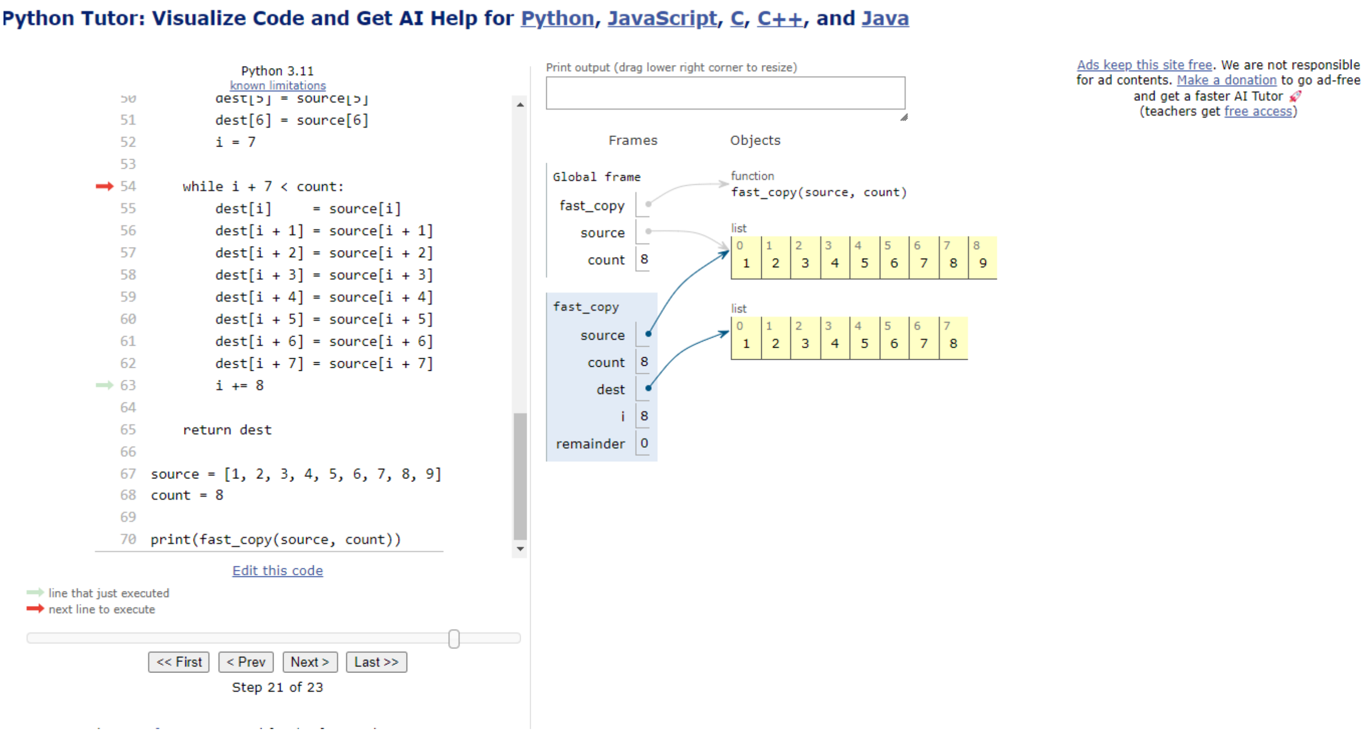
Task Write a Python function: fast_copy(source, count) that:

- Allocates a destination structure large enough to hold count elements.
- Determines how many complete transfer groups of eight elements are required.
- Performs any necessary initial element transfers using structured control flow rather than iteration.
- Completes the remaining transfers in a repetitive block that assigns exactly eight elements per iteration.
- Returns the destination structure containing the copied elements

Solutions with Test Driven Development Way:



```
def fast_copy(source, count):

    dest = [None] * count

    i = 0
    remainder = count % 8


    if remainder == 0:
        pass
    elif remainder == 1:
        dest[0] = source[0]
        i = 1
    elif remainder == 2:
        dest[0] = source[0]
        dest[1] = source[1]
        i = 2
    elif remainder == 3:
        dest[0] = source[0]
        dest[1] = source[1]
        dest[2] = source[2]
        i = 3
    elif remainder == 4:
        dest[0] = source[0]
        dest[1] = source[1]
        dest[2] = source[2]
        dest[3] = source[3]
        i = 4
    elif remainder == 5:
        dest[0] = source[0]
        dest[1] = source[1]
        dest[2] = source[2]
        dest[3] = source[3]
        dest[4] = source[4]
        i = 5
    elif remainder == 6:
        dest[0] = source[0]
        dest[1] = source[1]
        dest[2] = source[2]
        dest[3] = source[3]
        dest[4] = source[4]
        dest[5] = source[5]
        i = 6
    elif remainder == 7:
        dest[0] = source[0]
        dest[1] = source[1]
        dest[2] = source[2]
        dest[3] = source[3]
        dest[4] = source[4]
        dest[5] = source[5]
        dest[6] = source[6]
        i = 7

    while i + 7 < count:
        dest[i]     = source[i]
        dest[i + 1] = source[i + 1]
        dest[i + 2] = source[i + 2]
        dest[i + 3] = source[i + 3]
        dest[i + 4] = source[i + 4]
        dest[i + 5] = source[i + 5]
        dest[i + 6] = source[i + 6]
        dest[i + 7] = source[i + 7]
        i += 8

    return dest

source = [1, 2, 3, 4, 5, 6, 7, 8, 9]
count = 8

print(fast_copy(source, count))
```


Python Tutor: Visualize Code and Get AI Help for Python, JavaScript, C, C++, and Java

Python 3.11

```
        dest[5] = source[5]
        dest[6] = source[6]
        i = 7

    while i + 7 < count:
        dest[i]     = source[i]
        dest[i + 1] = source[i + 1]
        dest[i + 2] = source[i + 2]
        dest[i + 3] = source[i + 3]
        dest[i + 4] = source[i + 4]
        dest[i + 5] = source[i + 5]
        dest[i + 6] = source[i + 6]
        dest[i + 7] = source[i + 7]
        i += 8

    return dest

source = [1, 2, 3, 4, 5, 6, 7, 8, 9]
count = 8

print(fast_copy(source, count))
```

Global frame
fast_copy → function fast_copy(source, count)
source → list [1, 2, 3, 4, 5, 6, 7, 8, 9]
count 8

fast_copy
source → list [1, 2, 3, 4, 5, 6, 7, 8, 9]
count 8
dest → list [1, 2, 3, 4, 5, 6, 7, 8]
i 8
remainder 0

Step 21 of 23

My Learning about this fast copy:

I successfully implemented an optimized copy routine that handles leftover elements using explicit control flow and copies remaining data in fixed blocks of eight.

Through this task, I learned how loop unrolling and reduced branching improve performance in low-level, time-critical operations.