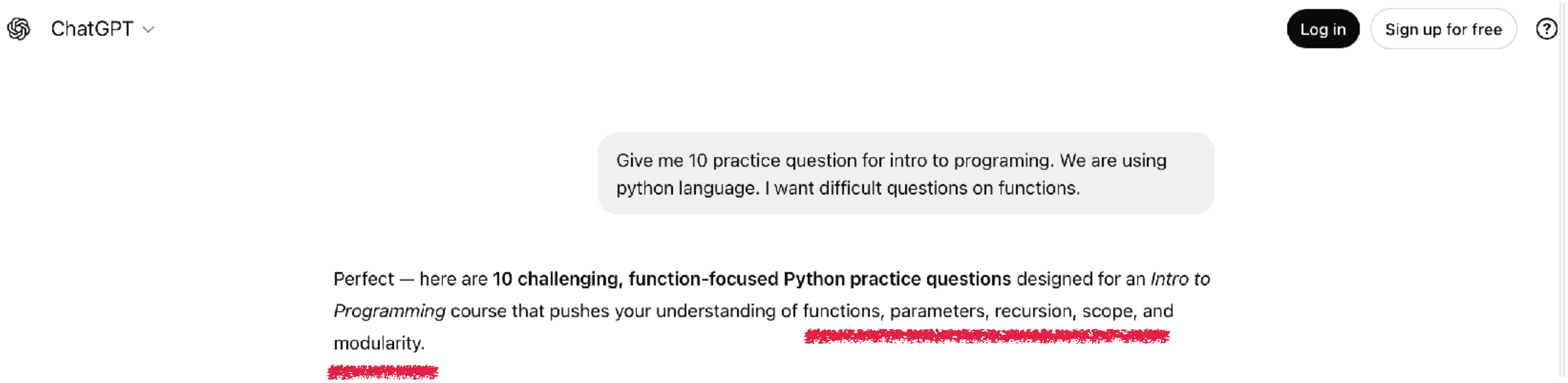
COL1000
Introduction to Programming

Priyanka Golia

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

This lecture was designed by an LLM and it is about LLMs



Give me 10 practice question for intro to programing. We are using python language. I want difficult questions on functions.

Perfect — here are **10 challenging, function-focused Python practice questions** designed for an *Intro to Programming* course that pushes your understanding of functions, parameters, recursion, scope, and modularity.

Have you ever used ChatGPT, Gemini, or Copilot?

What do you think it does when you ask it a question?

Do you trust its answers?

Have you ever used ChatGPT, Gemini, or Copilot?

What do you think it does when you ask it a question?

Do you trust its answers?

These tools are powered by Large Language Models (LLMs) — computer programs trained on huge amounts of text to generate language that sounds human. They don’t think or know facts like we do; they predict what words should come next, based on patterns learned from data

- LLMs are **pattern recognizers**, not knowledge holders.
- They are **trained on human-written text** — books, websites, conversations.
- They can generate useful, fluent, but sometimes incorrect or biased content.

- LLMs are **pattern recognizers**, not knowledge holders.
- They are **trained on human-written text** — books, websites, conversations.
- They can generate useful, fluent, but sometimes incorrect or biased content.

What an LLM can and can’t do:

- LLMs are **pattern recognizers**, not knowledge holders.
- They are **trained on human-written text** — books, websites, conversations.
- They can generate useful, fluent, but sometimes incorrect or biased content.

What an LLM can and can't do:

Can do:

Summarize information

Explain complex ideas simply

Help draft or brainstorm text

Translate or rewrite

Assist with coding, data, or logic

- LLMs are **pattern recognizers**, not knowledge holders.
- They are **trained on human-written text** — books, websites, conversations.
- They can generate useful, fluent, but sometimes incorrect or biased content.

What an LLM can and can't do:

Can do:

Summarize information

Explain complex ideas simply

Help draft or brainstorm text

Translate or rewrite

Assist with coding, data, or logic

Cannot do (or not reliably):

Verify truth or cite sources accurately

Access real-time or private information (unless connected to the web)

Understand emotions or intentions

How to interact with LLMs

Prompt Matters! —

A prompt is not just a question; it’s a set of instructions to guide the model.

How to interact with LLMs

Prompt Matters! —
A prompt is not just a question; it’s a set of instructions to guide the model.

You should use LLM as your co-creator, not as your substitute.

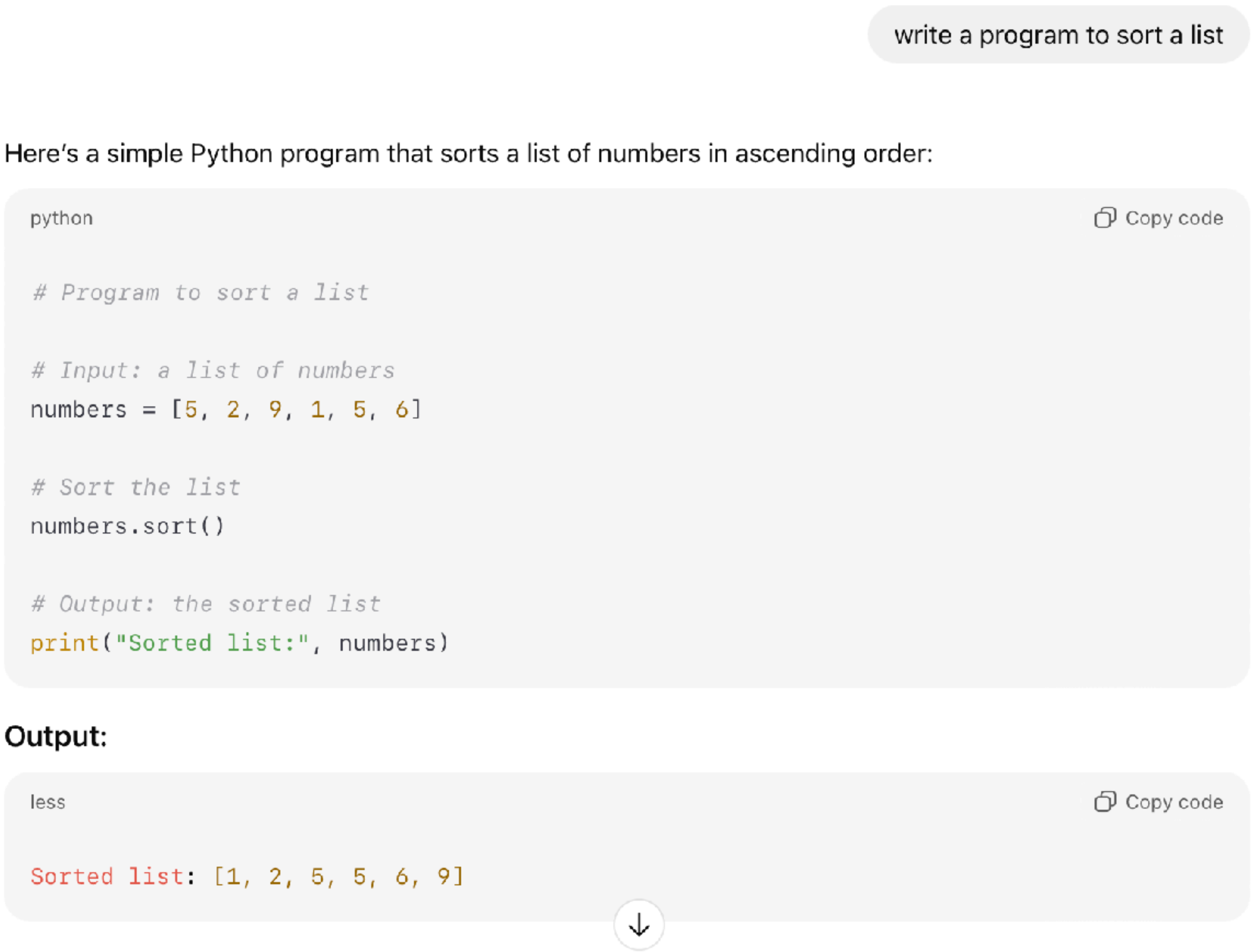
How to interact with LLMs

Prompt Matters! —

A prompt is not just a question; it’s a set of instructions to guide the model.

You should use LLM as your co-creator, not as your substitute.

Always verify what it returns — many many times its bugy



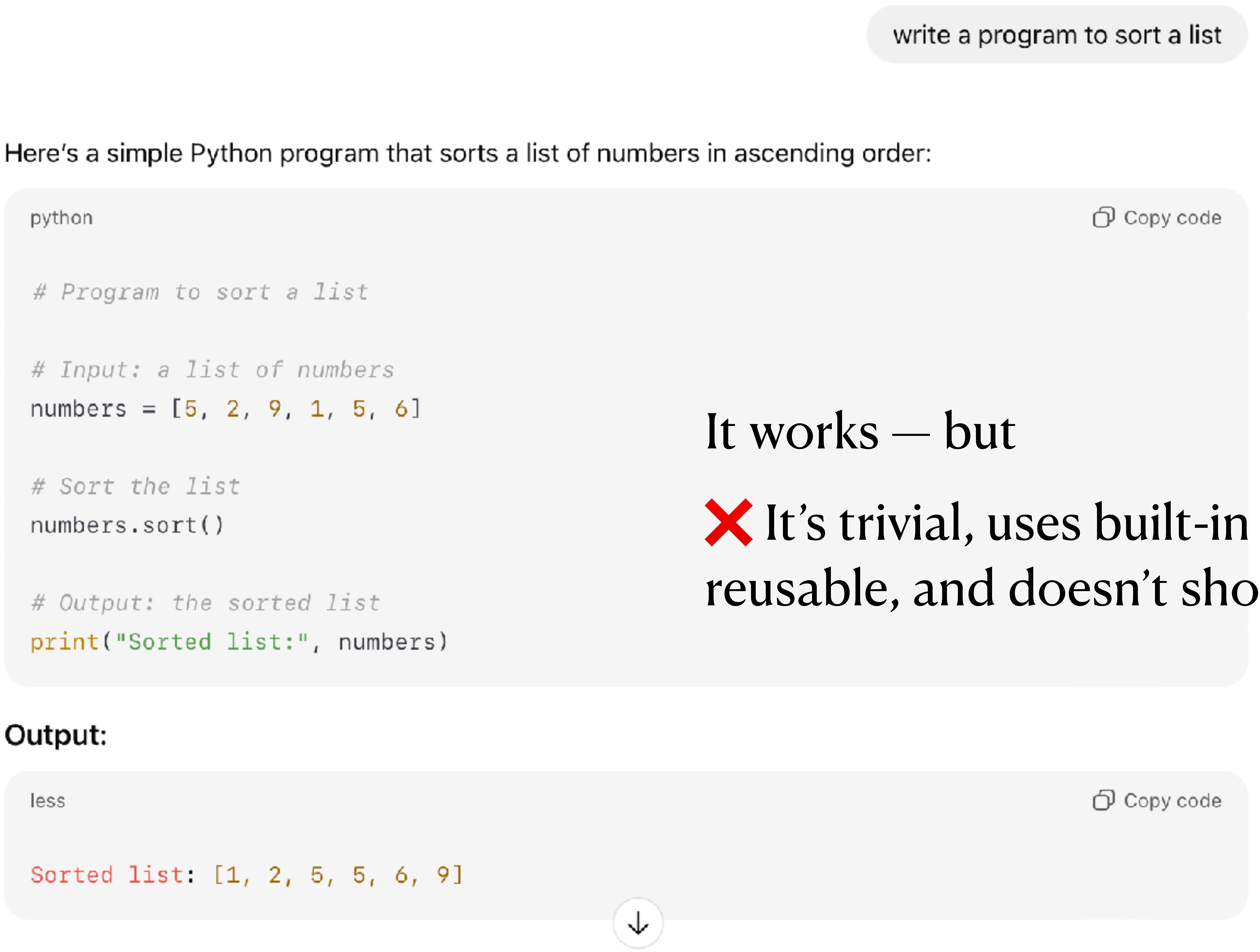
write a program to sort a list

Here's a simple Python program that sorts a list of numbers in ascending order:

```python
# Program to sort a list

# Input: a list of numbers
numbers = [5, 2, 9, 1, 5, 6]

# Sort the list
numbers.sort()

# Output: the sorted list
print("Sorted list:", numbers)
```

Output:

```less
Sorted list: [1, 2, 5, 5, 6, 9]
```



write a program to sort a list

Here's a simple Python program that sorts a list of numbers in ascending order:

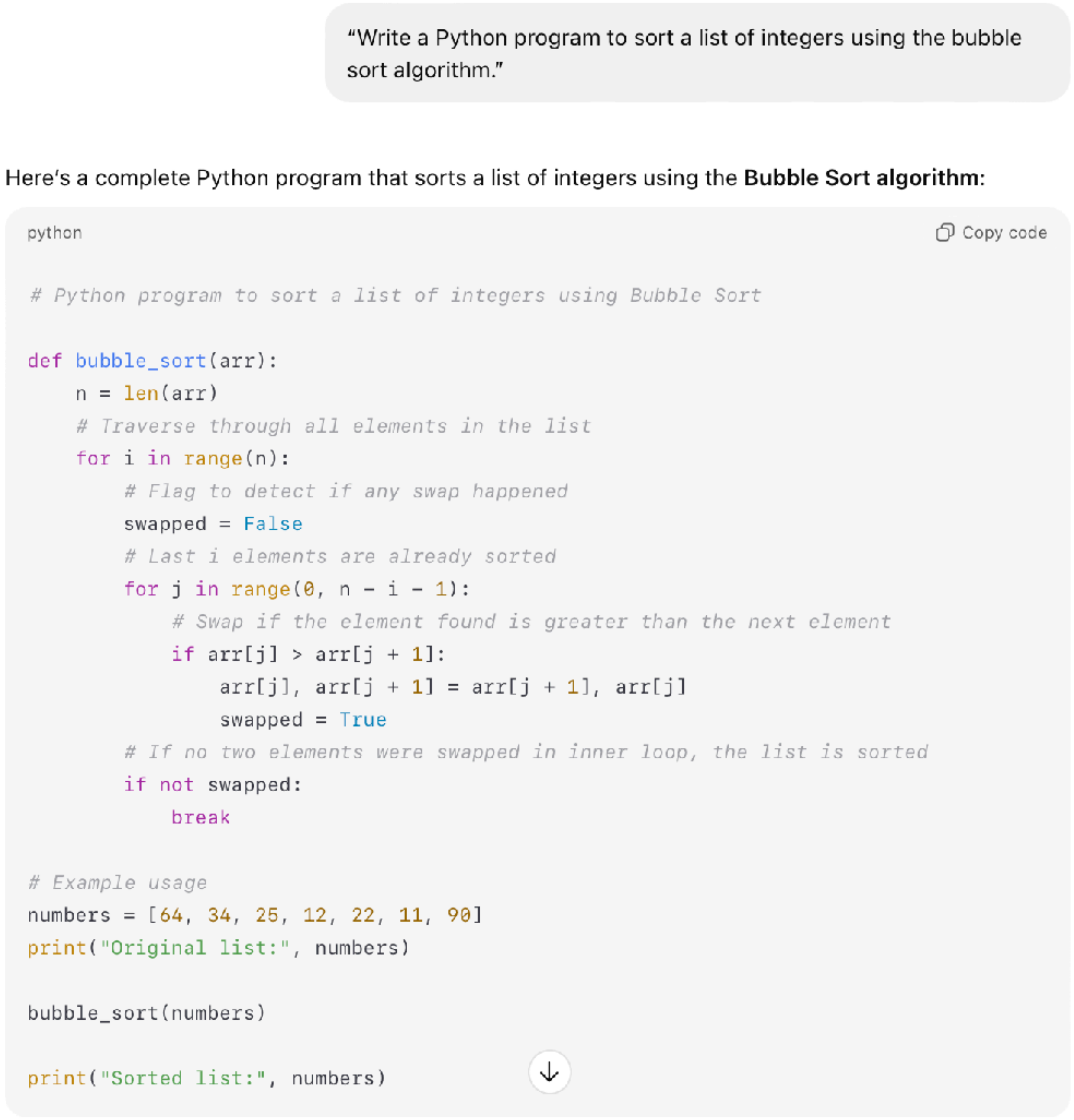
```python
# Program to sort a list

# Input: a list of numbers
numbers = [5, 2, 9, 1, 5, 6]

# Sort the list
numbers.sort()

# Output: the sorted list
print("Sorted list:", numbers)
```

Output:

```
Sorted list: [1, 2, 5, 5, 6, 9]
```

It works — but

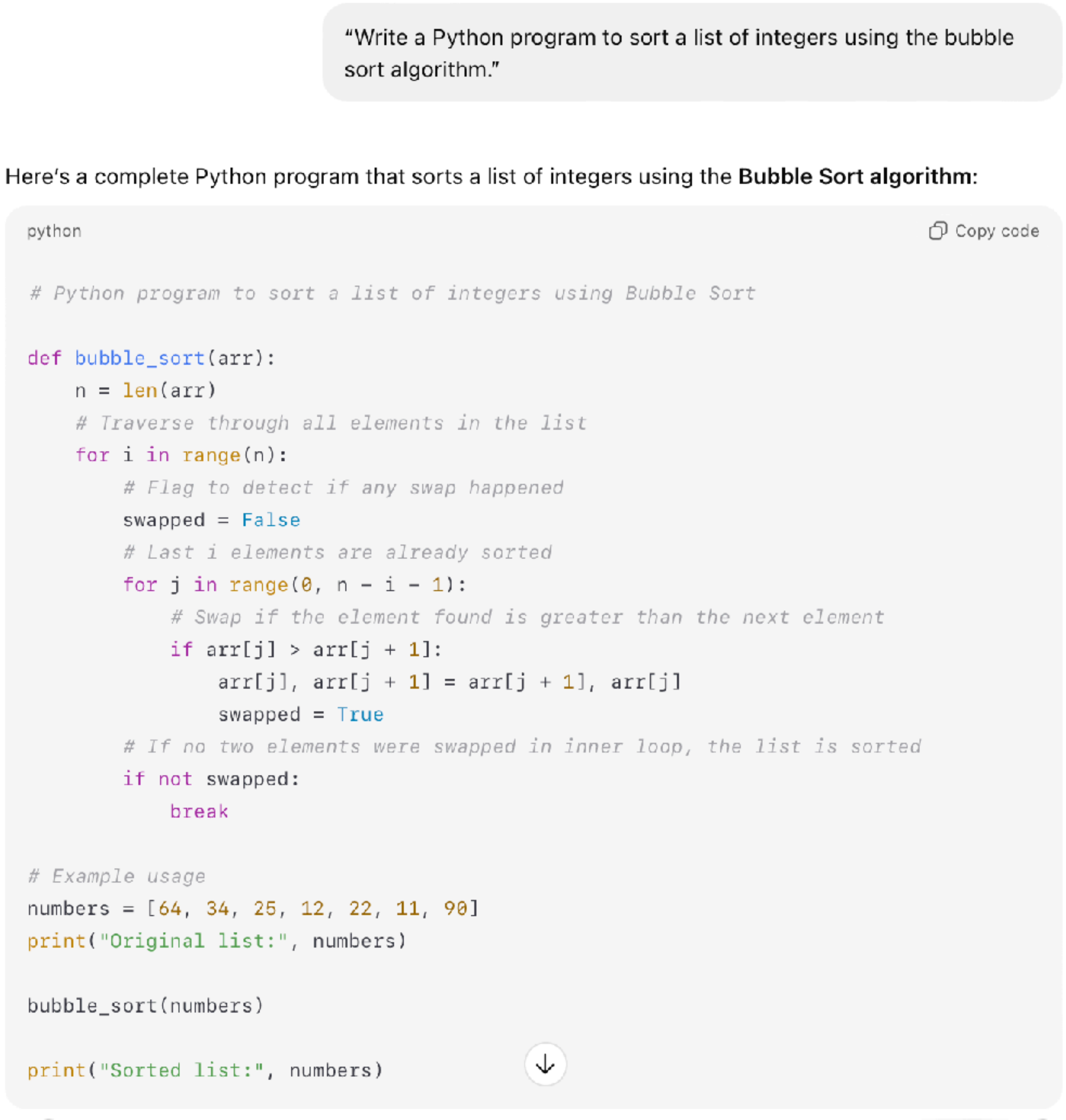
❌ It's trivial, uses built-in sort, no explanation, not reusable, and doesn't show understanding.



"Write a Python program to sort a list of integers using the bubble sort algorithm."

Here's a complete Python program that sorts a list of integers using the **Bubble Sort algorithm**:

```python
# Python program to sort a list of integers using Bubble Sort

def bubble_sort(arr):
    n = len(arr)
    # Traverse through all elements in the list
    for i in range(n):
        # Flag to detect if any swap happened
        swapped = False
        # Last i elements are already sorted
        for j in range(0, n - i - 1):
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # If no two elements were swapped in inner loop, the list is sorted
        if not swapped:
            break

# Example usage
numbers = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", numbers)

bubble_sort(numbers)

print("Sorted list:", numbers)
```



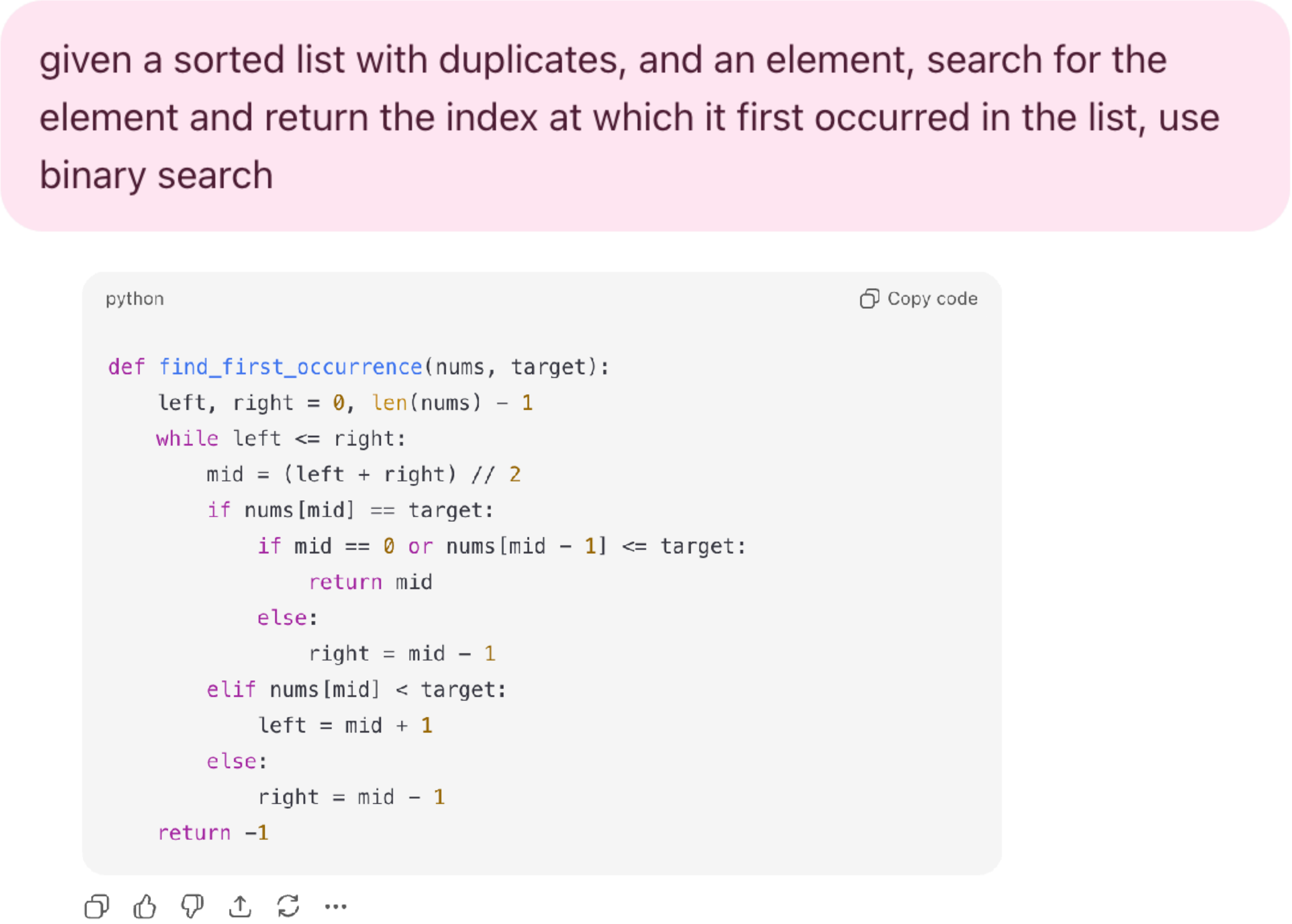
"Write a Python program to sort a list of integers using the bubble sort algorithm."

Here's a complete Python program that sorts a list of integers using the **Bubble Sort algorithm**:

```python
# Python program to sort a list of integers using Bubble Sort

def bubble_sort(arr):
    n = len(arr)
    # Traverse through all elements in the list
    for i in range(n):
        # Flag to detect if any swap happened
        swapped = False
        # Last i elements are already sorted
        for j in range(0, n - i - 1):
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # If no two elements were swapped in inner loop, the list is sorted
        if not swapped:
            break

# Example usage
numbers = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", numbers)

bubble_sort(numbers)

print("Sorted list:", numbers)
```

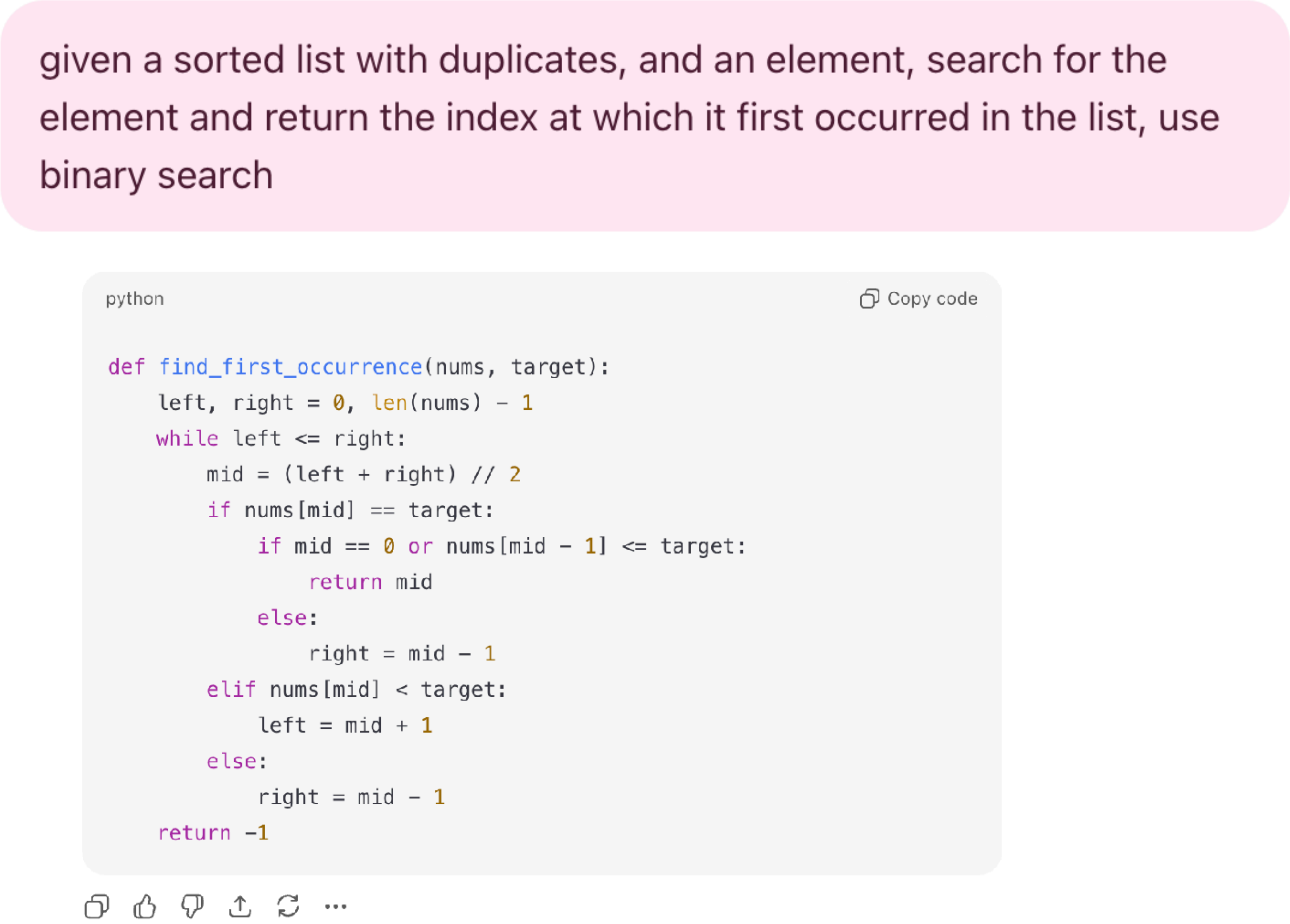
✅ Much better — readable, educational

given a sorted list with duplicates, and an element, search for the element and return the index at which it first occurred in the list, use binary search



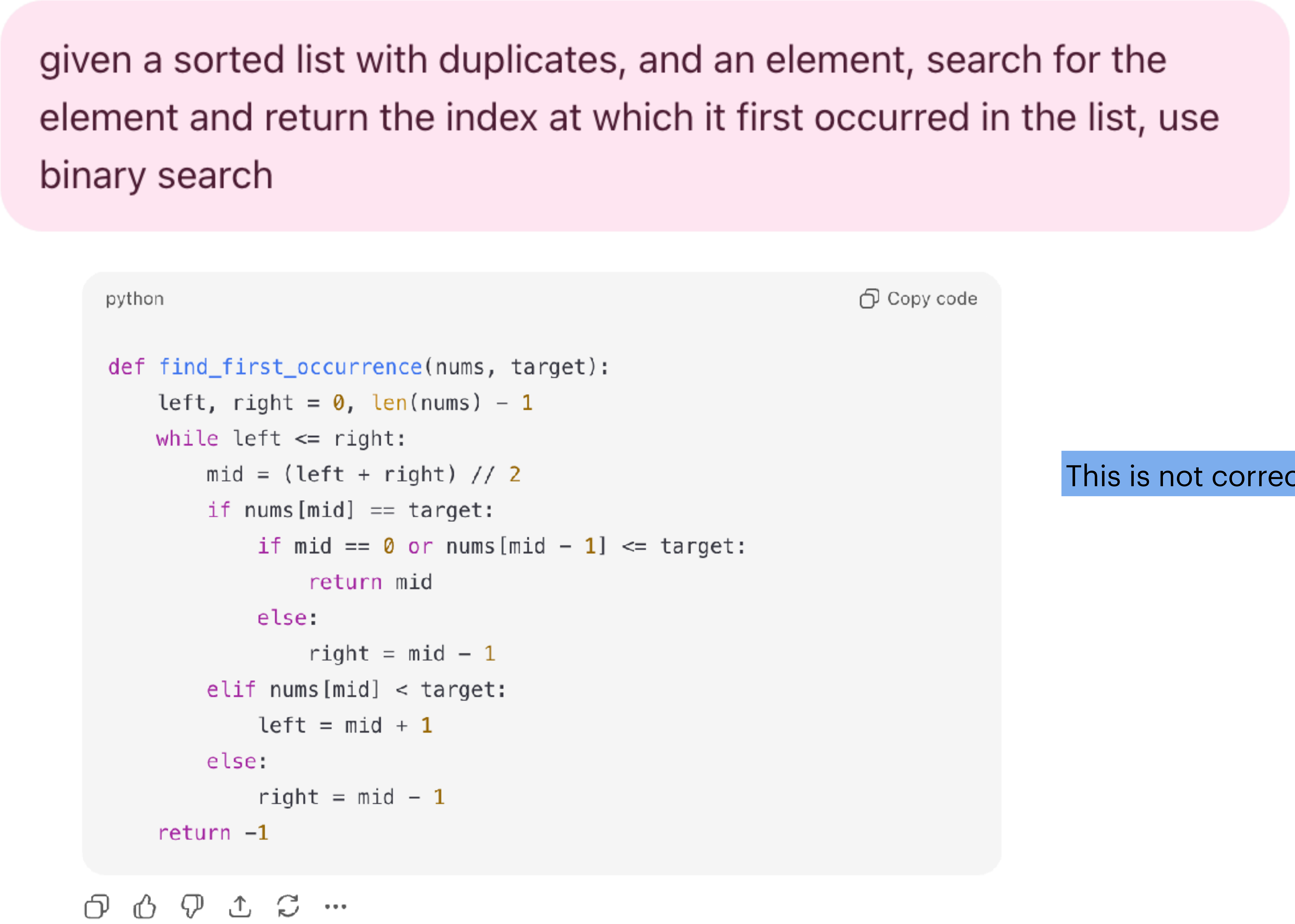
given a sorted list with duplicates, and an element, search for the element and return the index at which it first occurred in the list, use binary search

```python
def find_first_occurrence(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            if mid == 0 or nums[mid - 1] <= target:
                return mid
            else:
                right = mid - 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```



given a sorted list with duplicates, and an element, search for the element and return the index at which it first occurred in the list, use binary search

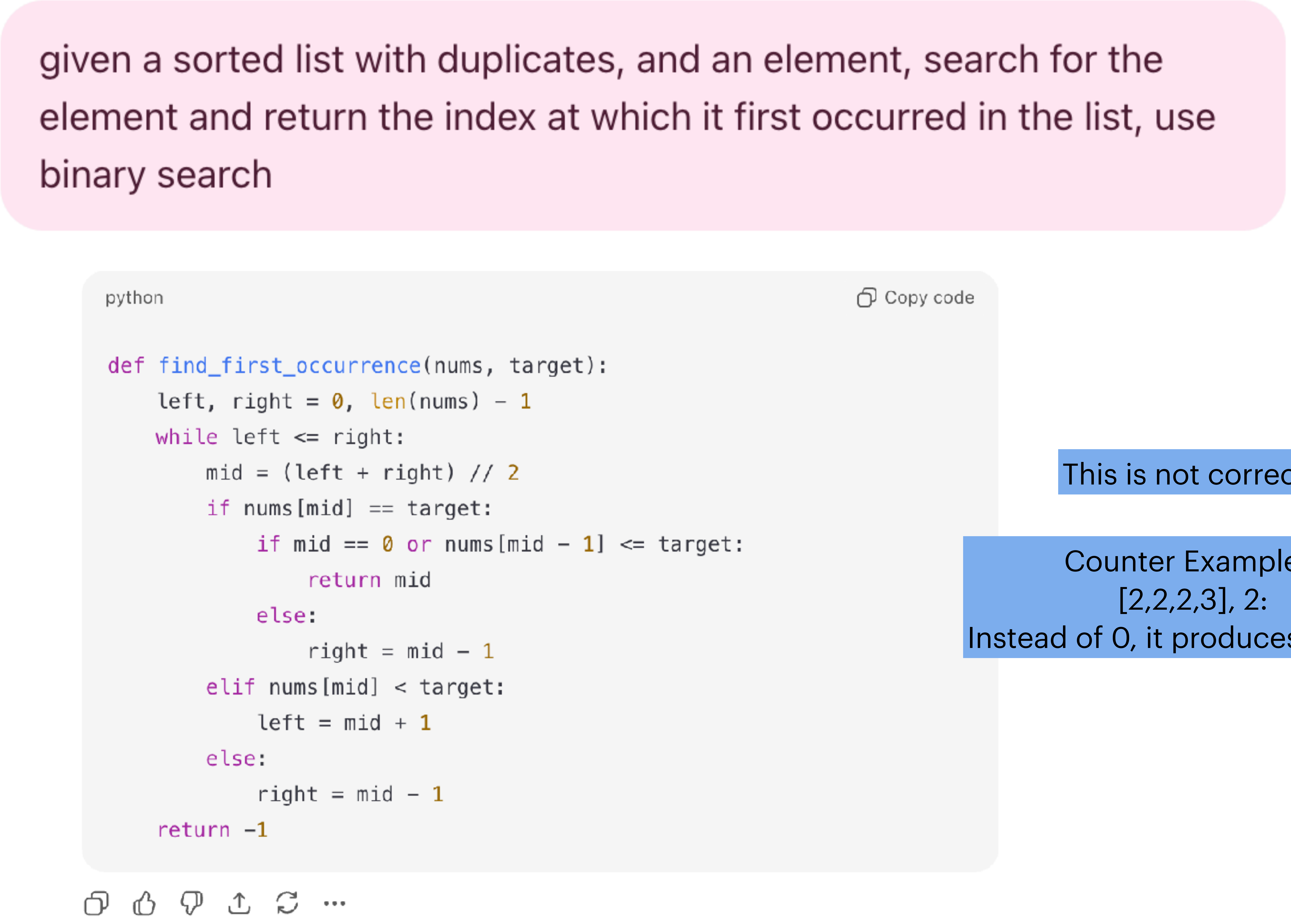
```python
def find_first_occurrence(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            if mid == 0 or nums[mid - 1] <= target:
                return mid
            else:
                right = mid - 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```



given a sorted list with duplicates, and an element, search for the element and return the index at which it first occurred in the list, use binary search

```python
def find_first_occurrence(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            if mid == 0 or nums[mid - 1] <= target:
                return mid
            else:
                right = mid - 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

This is not correct!!

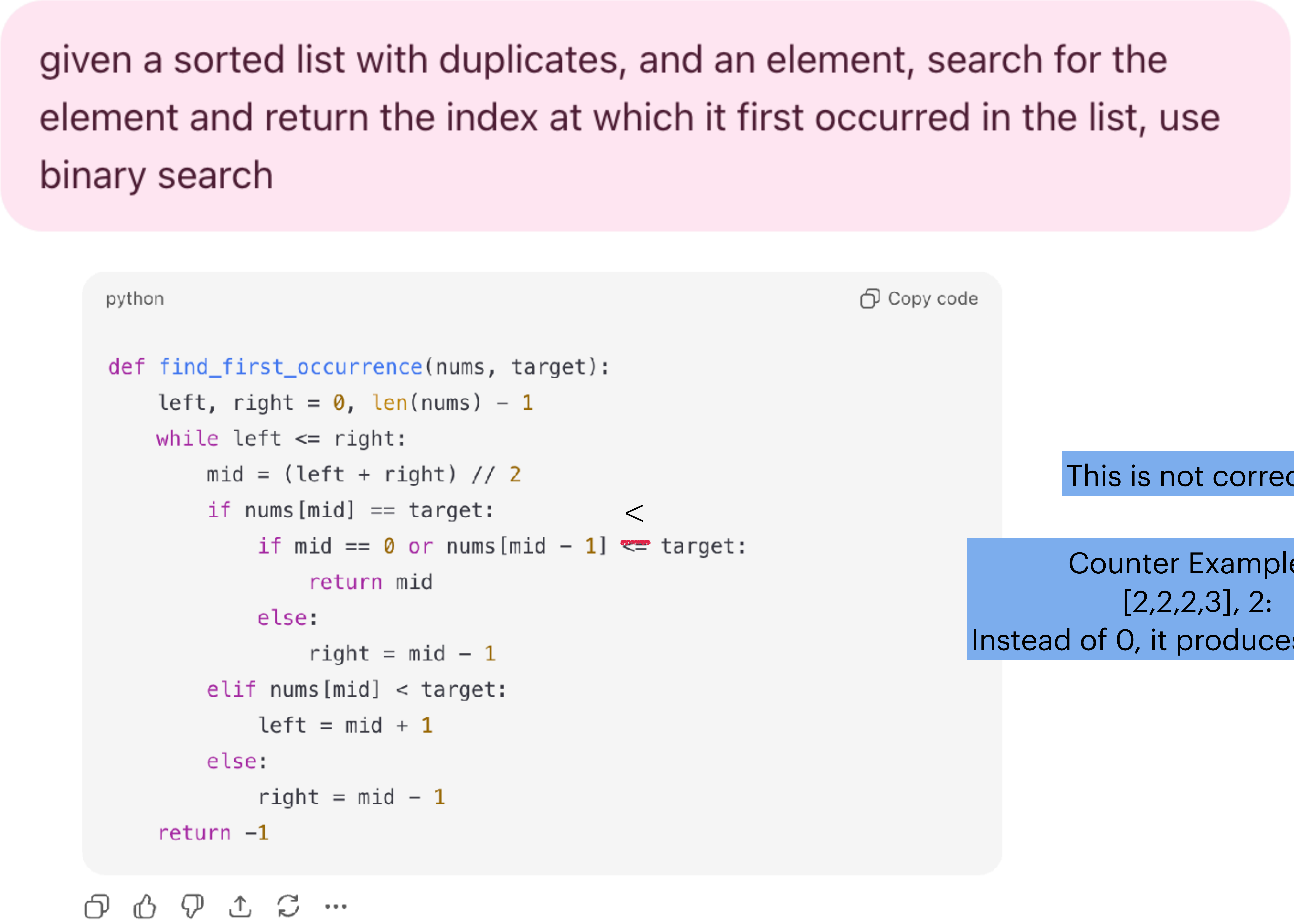


given a sorted list with duplicates, and an element, search for the element and return the index at which it first occurred in the list, use binary search

```python
def find_first_occurrence(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            if mid == 0 or nums[mid - 1] <= target:
                return mid
            else:
                right = mid - 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

This is not correct!!

Counter Example =
[2,2,2,3], 2:
Instead of 0, it produces output 1

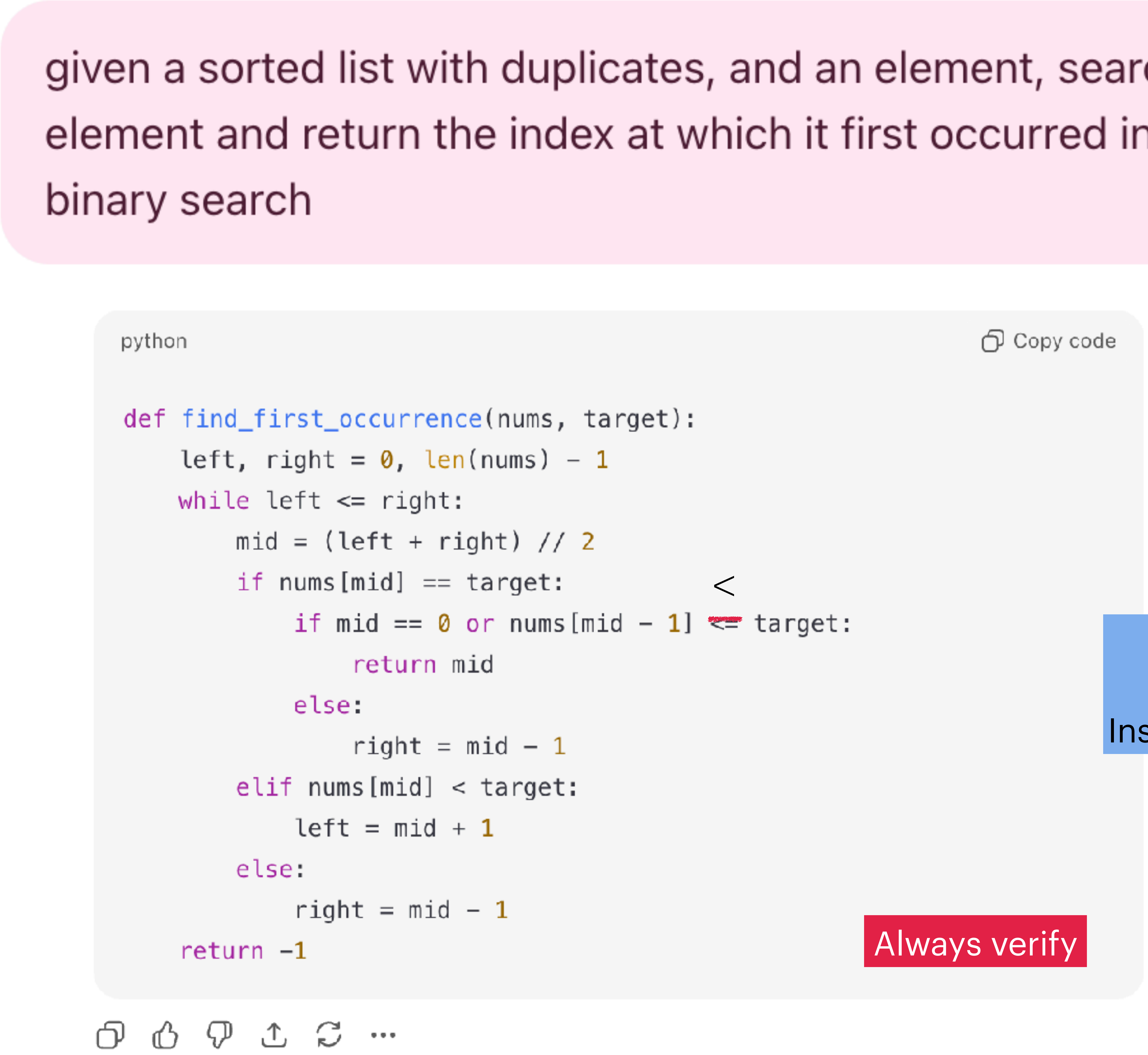


given a sorted list with duplicates, and an element, search for the element and return the index at which it first occurred in the list, use binary search

```python
def find_first_occurrence(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:                   <
            if mid == 0 or nums[mid - 1] <= target:
                return mid
            else:
                right = mid - 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

This is not correct!!

Counter Example =
[2,2,2,3], 2:
Instead of 0, it produces output 1



given a sorted list with duplicates, and an element, search for the element and return the index at which it first occurred in the list, use binary search

```python
def find_first_occurrence(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:                <
            if mid == 0 or nums[mid - 1] <= target:
                return mid
            else:
                right = mid - 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

This is not correct!!

Counter Example =
[2,2,2,3], 2:
Instead of 0, it produces output 1

Always verify

LLMs are tools, not teachers.

Your prompt shapes the result.

Refinement is part of the process.

Always verify and test.

Use it to learn, not to skip learning.

LLMs are tools, not teachers.

They can *generate* ideas and code, but they do not *understand* them. Always think before accepting their output.

Your prompt shapes the result.

Refinement is part of the process.

Always verify and test.

Use it to learn, not to skip learning.

LLMs are tools, not teachers.

They can *generate* ideas and code, but they do not *understand* them. Always think before accepting their output.

Your prompt shapes the result.

The more clearly you describe what you want — language, constraints, examples, explanations — the more relevant the response will be.

Refinement is part of the process.

Always verify and test.

Use it to learn, not to skip learning.

LLMs are tools, not teachers.

They can *generate* ideas and code, but they do not *understand* them. Always think before accepting their output.

Your prompt shapes the result.

The more clearly you describe what you want — language, constraints, examples, explanations — the more relevant the response will be.

Refinement is part of the process.

Using LLMs effectively means *iterating*: trying, reviewing, and improving your prompt based on what you get.

Always verify and test.

Use it to learn, not to skip learning.

LLMs are tools, not teachers.

They can *generate* ideas and code, but they do not *understand* them. Always think before accepting their output.

Your prompt shapes the result.

The more clearly you describe what you want — language, constraints, examples, explanations — the more relevant the response will be.

Refinement is part of the process.

Using LLMs effectively means *iterating*: trying, reviewing, and improving your prompt based on what you get.

Always verify and test.

Code produced by an LLM may compile but still be inefficient or incorrect. Run it, test it, and reason about it.

Use it to learn, not to skip learning.

LLMs are tools, not teachers.

They can *generate* ideas and code, but they do not *understand* them. Always think before accepting their output.

Your prompt shapes the result.

The more clearly you describe what you want — language, constraints, examples, explanations — the more relevant the response will be.

Refinement is part of the process.

Using LLMs effectively means *iterating*: trying, reviewing, and improving your prompt based on what you get.

Always verify and test.

Code produced by an LLM may compile but still be inefficient or incorrect. Run it, test it, and reason about it.

Use it to learn, not to skip learning.

LLMs are valuable for understanding concepts, debugging, or exploring alternatives. But relying on them without comprehension limits growth.