Tutorial 2

Pseudocode styles and interview workflow

Jan Bures

18YZALG – Basics of Algorithmization, Summer Semester 2026

Today

- Quick reminder: what pseudocode is (from Tutorial 1)
- A **gallery of pseudocode styles**: same ideas, different looks
- Interview scene + a standardized workflow for explaining solutions
- Final recap: what you now have from **Tutorial 1 + 2**

Reminder: what pseudocode is

Pseudocode (from Tutorial 1)

A **language-independent** description of an algorithm. It is meant to be **readable** and **unambiguous**, not necessarily runnable.

A minimal shape (any style is OK if consistent)

- A clear name and inputs
- Standard control flow (`if/else`, `for/while`, `return`)
- Indentation shows structure
- Explicit `return` paths (no mystery ending)

Reminder: what pseudocode is

Pseudocode (from Tutorial 1)

A **language-independent** description of an algorithm. It is meant to be **readable** and **unambiguous**, not necessarily runnable.

A minimal shape (any style is OK if consistent)

- A clear name and inputs
- Standard control flow (`if/else`, `for/while`, `return`)
- Indentation shows structure
- Explicit `return` paths (no mystery ending)

Reminder: what pseudocode is

Pseudocode (from Tutorial 1)

A **language-independent** description of an algorithm. It is meant to be **readable** and **unambiguous**, not necessarily runnable.

A minimal shape (any style is OK if consistent)

- A clear name and inputs
- Standard control flow (`if/else`, `for/while`, `return`)
- Indentation shows structure
- Explicit `return` paths (no mystery ending)

Reminder: what pseudocode is

Pseudocode (from Tutorial 1)

A **language-independent** description of an algorithm. It is meant to be **readable** and **unambiguous**, not necessarily runnable.

A minimal shape (any style is OK if consistent)

- A clear name and inputs
- Standard control flow (`if/else`, `for/while`, `return`)
- Indentation shows structure
- Explicit `return` paths (no mystery ending)

Reminder: what pseudocode is

Pseudocode (from Tutorial 1)

A **language-independent** description of an algorithm. It is meant to be **readable** and **unambiguous**, not necessarily runnable.

A minimal shape (any style is OK if consistent)

- A clear name and inputs
- Standard control flow (`if/else`, `for/while`, `return`)
- Indentation shows structure
- Explicit `return` paths (no mystery ending)

Pseudocode styles: it is a dial, not a rule

- **Audience:** you / teammate / interviewer / examiner
- **Detail level:** high-level steps ↔ almost code
- **Notation:** indices vs `for each`, temporary variables vs direct returns
- **Goal stays the same:** someone else can implement it without guessing

Takeaway

Pick a style that matches the situation, then be consistent.

Pseudocode styles: it is a dial, not a rule

- **Audience:** you / teammate / interviewer / examiner
- **Detail level:** high-level steps ↔ almost code
- **Notation:** indices vs `for each`, temporary variables vs direct returns
- **Goal stays the same:** someone else can implement it without guessing

Takeaway

Pick a style that matches the situation, then be consistent.

Pseudocode styles: it is a dial, not a rule

- **Audience:** you / teammate / interviewer / examiner
- **Detail level:** high-level steps ↔ almost code
- **Notation:** indices vs `for each`, temporary variables vs direct returns
- **Goal stays the same:** someone else can implement it without guessing

Takeaway

Pick a style that matches the situation, then be consistent.

Pseudocode styles: it is a dial, not a rule

- **Audience:** you / teammate / interviewer / examiner
- **Detail level:** high-level steps ↔ almost code
- **Notation:** indices vs `for each`, temporary variables vs direct returns
- **Goal stays the same:** someone else can implement it without guessing

Takeaway

Pick a style that matches the situation, then be consistent.

Pseudocode styles: it is a dial, not a rule

- **Audience:** you / teammate / interviewer / examiner
- **Detail level:** high-level steps ↔ almost code
- **Notation:** indices vs `for each`, temporary variables vs direct returns
- **Goal stays the same:** someone else can implement it without guessing

Takeaway

Pick a style that matches the situation, then be consistent.

Pseudocode styles: it is a dial, not a rule

- **Audience:** you / teammate / interviewer / examiner
- **Detail level:** high-level steps ↔ almost code
- **Notation:** indices vs `for each`, temporary variables vs direct returns
- **Goal stays the same:** someone else can implement it without guessing

Takeaway

Pick a style that matches the situation, then be consistent.

Style 1: structured template

Pseudocode

```
Problem PREFIX-SUMS
Input: list A of n numbers
Output: list B where B[i] = A[0] + ... + A[i]

Algorithm PREFIX-SUMS(A):
    n = len(A)
    B = new list of length n
    s = 0
    for i in 0..n-1:
        s = s + A[i]
        B[i] = s
    return B
```

Style 2: Python-like pseudocode

Pseudocode

```
def prefix_sums(A):
    B = []
    s = 0
    for x in A:
        s += x
        B.append(s)
    return B
```

What this style communicates

You are close to an implementation; details like append are explicit.

Style 2: Python-like pseudocode

Pseudocode

```
def prefix_sums(A):
    B = []
    s = 0
    for x in A:
        s += x
        B.append(s)
    return B
```

What this style communicates

You are close to an implementation; details like `append` are explicit.

Style 3: numbered recipe steps

Pseudocode

```
Algorithm FILTER-NONNEGATIVE(A):
    1. Create empty list R
    2. For each x in A:
           if x >= 0:
               append x to R
    3. Return R
```

What this style communicates

The algorithm is a **procedure** with an order of steps.

Style 3: numbered recipe steps

Pseudocode

```
Algorithm FILTER-NONNEGATIVE(A):
    1. Create empty list R
    2. For each x in A:
           if x >= 0:
               append x to R
    3. Return R
```

What this style communicates

The algorithm is a **procedure** with an order of steps.

Style 4: guard clauses + explicit returns

Pseudocode

```
Algorithm SAFE-AVERAGE(A):
    n = len(A)
    if n == 0:
        return None

    s = 0
    for x in A:
        s = s + x
    return s / n
```

Why this is useful

Readers immediately see what happens in edge cases and where the algorithm ends.

Style 4: guard clauses + explicit returns

Pseudocode

```
Algorithm SAFE-AVERAGE(A):
    n = len(A)
    if n == 0:
        return None

    s = 0
    for x in A:
        s = s + x
    return s / n
```

Why this is useful

Readers immediately see what happens in edge cases and where the algorithm ends.

Style 5: indices vs `for each`

Index-explicit

```
Algorithm COUNT-ZEROS(A):
    n = len(A)
    c = 0
    for i in 0..n-1:
        if A[i] == 0:
            c = c + 1
    return c
```

For-each

```
Algorithm COUNT-ZEROS(A):
    c = 0
    for x in A:
        if x == 0:
            c = c + 1
    return c
```

Choosing a style quickly

- **Whiteboard / interview:** prefer compact, code-like pseudocode (easy to translate).
- **Exam / homework write-up:** structured template with edge-cases and language-independent content is great (+clear inputs/outputs).
- **Team discussion:** sometimes numbered steps + a picture is fastest.
- If in doubt: **write the returns and bounds first**, then fill the loop body.

Choosing a style quickly

- **Whiteboard / interview:** prefer compact, code-like pseudocode (easy to translate).
- **Exam / homework write-up:** structured template with edge-cases and language-independent content is great (+clear inputs/outputs).
- **Team discussion:** sometimes numbered steps + a picture is fastest.
- If in doubt: **write the returns and bounds first**, then fill the loop body.

Choosing a style quickly

- **Whiteboard / interview:** prefer compact, code-like pseudocode (easy to translate).
- **Exam / homework write-up:** structured template with edge-cases and language-independent content is great (+clear inputs/outputs).
- **Team discussion:** sometimes numbered steps + a picture is fastest.
- If in doubt: **write the returns and bounds first**, then fill the loop body.

Choosing a style quickly

- **Whiteboard / interview:** prefer compact, code-like pseudocode (easy to translate).
- **Exam / homework write-up:** structured template with edge-cases and language-independent content is great (+clear inputs/outputs).
- **Team discussion:** sometimes numbered steps + a picture is fastest.
- If in doubt: **write the returns and bounds first**, then fill the loop body.

What to remember?

One sentence to keep

If your pseudocode is easy to read, your solution is easy to trust.

Interview scene: what is actually happening

- You are solving a problem **and** showing how you think.
- The interviewer sees only what you **say** and what you **write**.
- Pseudocode is your shared language: fast to write, precise enough to implement.
- The easiest failure mode: jumping into code before agreeing on the task.

Interview scene: what is actually happening

- You are solving a problem **and** showing how you think.
- The interviewer sees only what you **say** and what you **write**.
- Pseudocode is your shared language: fast to write, precise enough to implement.
- The easiest failure mode: jumping into code before agreeing on the task.

Interview scene: what is actually happening

- You are solving a problem **and** showing how you think.
- The interviewer sees only what you **say** and what you **write**.
- Pseudocode is your shared language: fast to write, precise enough to implement.
- The easiest failure mode: jumping into code before agreeing on the task.

Interview scene: what is actually happening

- You are solving a problem **and** showing how you think.
- The interviewer sees only what you **say** and what you **write**.
- Pseudocode is your shared language: fast to write, precise enough to implement.
- The easiest failure mode: jumping into code before agreeing on the task.

A standardized interview workflow

1. **Restate** the task in your own words (one sentence).
2. **Clarify** terms + inputs/outputs (what is returned? what about empty input?).
3. **Plan** the approach out loud (one paragraph, no code yet).
4. **Write** pseudocode with explicit control flow and returns.
5. **Walk through** one tiny example (to catch misunderstandings).
6. **Wrap up** with a short recap (what it returns, when it stops).

A standardized interview workflow

1. **Restate** the task in your own words (one sentence).
2. **Clarify** terms + inputs/outputs (what is returned? what about empty input?).
3. **Plan** the approach out loud (one paragraph, no code yet).
4. **Write** pseudocode with explicit control flow and returns.
5. **Walk through** one tiny example (to catch misunderstandings).
6. **Wrap up** with a short recap (what it returns, when it stops).

A standardized interview workflow

1. **Restate** the task in your own words (one sentence).
2. **Clarify** terms + inputs/outputs (what is returned? what about empty input?).
3. **Plan** the approach out loud (one paragraph, no code yet).
4. **Write** pseudocode with explicit control flow and returns.
5. **Walk through** one tiny example (to catch misunderstandings).
6. **Wrap up** with a short recap (what it returns, when it stops).

A standardized interview workflow

1. **Restate** the task in your own words (one sentence).
2. **Clarify** terms + inputs/outputs (what is returned? what about empty input?).
3. **Plan** the approach out loud (one paragraph, no code yet).
4. **Write** pseudocode with explicit control flow and returns.
5. **Walk through** one tiny example (to catch misunderstandings).
6. **Wrap up** with a short recap (what it returns, when it stops).

A standardized interview workflow

1. **Restate** the task in your own words (one sentence).
2. **Clarify** terms + inputs/outputs (what is returned? what about empty input?).
3. **Plan** the approach out loud (one paragraph, no code yet).
4. **Write** pseudocode with explicit control flow and returns.
5. **Walk through** one tiny example (to catch misunderstandings).
6. **Wrap up** with a short recap (what it returns, when it stops).

A standardized interview workflow

1. **Restate** the task in your own words (one sentence).
2. **Clarify** terms + inputs/outputs (what is returned? what about empty input?).
3. **Plan** the approach out loud (one paragraph, no code yet).
4. **Write** pseudocode with explicit control flow and returns.
5. **Walk through** one tiny example (to catch misunderstandings).
6. **Wrap up** with a short recap (what it returns, when it stops).

Mini-demo: `FIRST-VIOLATION` (pseudocode)

Task

Return the first index i where $A[i] \notin [lo, hi]$. If all values are inside, return -1.

Pseudocode

```
Algorithm FIRST-VIOLATION(A, lo, hi):
    for i in 0..len(A)-1:
        if A[i] < lo or A[i] > hi:
            return i
    return -1
```

Mini-demo: `FIRST-VIOLATION` (pseudocode)

Task

Return the first index i where $A[i] \notin [lo, hi]$. If all values are inside, return -1.

Pseudocode

```
Algorithm FIRST-VIOLATION(A, lo, hi):
    for i in 0..len(A)-1:
        if A[i] < lo or A[i] > hi:
            return i
    return -1
```

Mini-demo: what you say while writing

A compact spoken script (you can reuse it)

Restate: “We need the first position where the value is outside $[lo, hi]$, otherwise -1.”

Clarify: “If the list is empty, we return -1.”

Plan: “I will scan from left to right and return immediately on the first violation.”

Walkthrough: “On $[2, 5, 7]$ with $[1, 6]$, we return index 2.”

Wrap: “If we finish the loop, there was no violation, so we return -1.”

Recap: what you have from Tutorial 1 + 2

Tutorial 1

- What an algorithm is + how to describe it
- Pseudocode as a clear procedure
- Correctness & complexity as the two questions to ask

Tutorial 2

- Pseudocode can look different: pick a style and stay consistent
- Show the algorithm with explicit control flow and returns
- Interview workflow: restate → clarify → plan → pseudocode → walkthrough → wrap

Recap: what you have from Tutorial 1 + 2

Tutorial 1

- What an algorithm is + how to describe it
- Pseudocode as a clear procedure
- Correctness & complexity as the two questions to ask

Tutorial 2

- Pseudocode can look different: pick a style and stay consistent
- Show the algorithm with explicit control flow and returns
- Interview workflow: restate → clarify → plan → pseudocode → walkthrough → wrap