

FUNDAMENTALS OF AI-AUGMENTED ENGINEERING: A CONVERSATIONAL APPROACH

Taner Onsay, Ph.D. & Sebnem Onsay
Authors
August 2026 · Revision 1.0

© 2026 AI First Engineering

Problem



Model



Compute



Analyze



Validate



Improve

*The engineering workflow you will use throughout this book
by Taner & Sebnem Onsay,*



CONTENTS OF CHAPTER 2:

Chapter 1 asked: how does an engineer solve a problem? **Chapter 2 asks:** how do we express that thinking precisely enough that a computer can execute it?

1

From engineering procedure to algorithm

The extremely literal assistant · anatomy of an algorithm · pseudocode · running it by hand



2

Decisions & repetition

Cargo hoist · IF / ELSE · FOR vs WHILE · a loop must know when to stop



3

The engineering search — vehicle stopping distance

Excel makes the repetition visible · pseudocode captures it · Python automates it



4

Functions, decomposition & the AI coding companion

Named, testable engineering tasks · THINK › PSEUDOCODE › AI › CODE



5

Test · Debug · Validate

RUNS ≠ CORRECT · evidence-first debugging · BREAK THE AI · a disagreement worth having



Prediction before computation is used throughout the chapter.

CHAPTER - 2

PART 1 · ≈ 65 MIN

From Engineering Procedure to Algorithm

You already solve problems in steps. Now we make those steps explicit enough for a machine to follow.

1

Problem

>

Model

>

Compute

>

Analyze

>

Validate

>

Improve



You already lived the workflow

In the ball-drop investigation of Chapter 1 you...

-  predicted the fall time before measuring it,
-  measured it several independent ways (video, tracking, timing),
-  compared model against experiment and confronted disagreement,
-  used AI in conversation — and made the final call yourself.

Evidence first → explanation second. That habit carries straight into this chapter.

THE CHAPTER 1 MODEL

$$h = \frac{1}{2} g t^2$$

solve for t:

$$t = \sqrt{(2h / g)}$$

Keep this formula in view. It is about to become your first algorithm — and, later in this chapter, your first reusable function.

Rule of engagement for this chapter: AI assists. Human verifies. Engineer decides.

Problem



Model



Compute



Analyze



Validate



Improve



The extremely literal assistant

Imagine an assistant who...



Does exactly what you say

Every instruction is followed to the letter — no more, no less.



Assumes nothing

No common sense, no context, no "you know what I mean."



Never gets tired

Will happily repeat a step a million times — or forever.



Cannot ask what you meant

Ambiguity does not get resolved. It gets executed.

YOUR TASK

Explain to this assistant, precisely, how to compute

$$t = \sqrt{(2h / g)}$$

What must the assistant be told?

- What information does it receive?
- What does it already need to know?
- What exactly does it do — in what order?
- What does it hand back?

Talk to your neighbor — 2 minutes. Write the instructions as numbered steps.



Even a one-line formula hides a procedure



Input → known constant → ordered calculation → output. You have just written an algorithm.

ALGORITHM

An explicit, step-by-step computational procedure for solving a problem — precise enough that nothing is left to interpretation.

A program is simply an algorithm written in a language a computer can execute. The hard part is not the language — it is the thinking.





Five questions that expose any algorithm



INPUTS

What information does the procedure receive?

e.g. drop height h



PROCESS

What calculations happen — in what order?

e.g. $2h/g$, then v



DECISIONS

Where can the path split, based on a condition?

e.g. is h valid?



REPETITION

What gets done again and again — and when does it stop?

e.g. several heights?



OUTPUTS

What result is reported at the end?

e.g. fall time t

Decisions and repetition are the two new ideas of this chapter. Chapter 1 procedures ran straight through. Real engineering procedures branch and repeat — and this chapter's engineering problems will force both to appear.

Problem



Model



Compute



Analyze



Validate



Improve



You have been running algorithms all your life

A RECIPE

Chocolate-chip cookies

1. Mix butter and sugar
2. Add the egg, mix
3. Stir in flour, chocolate
4. Bake at 350 °F for 8 min

INPUTS · PROCESS · OUTPUT

Ingredients in, cookies out, steps in strict order.

A RESTAURANT HOST

Seating the next party

IF party of 1 → counter

ELSE IF party of 2 → small table

ELSE → large table

DECISIONS

A decision chain, run in the head all night long.

Ask the five questions of either one — they answer. *Two more, from the world of machines, on the next slide.*

Problem



Model



Compute



Analyze



Validate



Improve



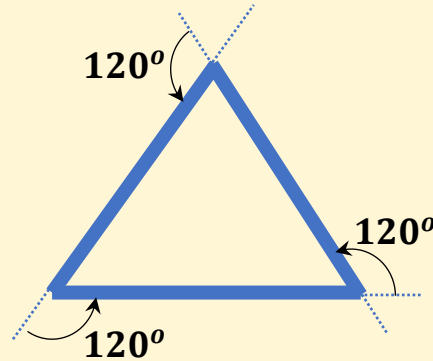
Machines run them too — the same five questions

A DRAWING ROBOT

A pen that draws a triangle

Pen down

```
3 × : forward 100,  
      turn left 120
```



*Three identical passes
— a known count.*

REPETITION

A THERMOSTAT

Hold the room at 20 °C

Read the temperature

Too cold → heat on

Warm enough → heat off

... check again, forever

DECISION + REPETITION

*A decision inside a
repetition.*

An algorithm is not a computer thing. People have always run them. What a computer adds is only a demand: every step written down, with nothing left to common sense.

Problem



Model



Compute



Analyze



Validate



Improve



An algorithm you already run: making toast

ORDINARY LANGUAGE

“Put a slice of bread in the toaster and take it out when it’s done.”

Any person can do this. The literal assistant cannot: which setting? how long is “done”? what if it comes out pale? and what, exactly, is handed back?

WHAT THE ALGORITHM ADDED

A named input and output. bread in, toast out — nothing is assumed.

A known setting, stated. dial = 3, the way $g = 9.81$ will be.

A stop for the waiting. “until the lever pops up” — repetition with a stopping rule.

A decision, with both outcomes. too pale → toast again; otherwise carry on.

THE ALGORITHM — EVERY STEP EXPLICIT

making toast — precise, complete, in order

1. Take one slice of bread
2. Set the toaster dial to 3
3. Put the bread in the toaster
4. Press the lever down
5. Wait until the lever pops up
6. If the toast is too pale,
press and wait once more
7. Otherwise, carry on
8. Serve the toast on a plate

INPUT

KNOWN SETTING

PROCESS

PROCESS

REPETITION · STOPS

DECISION

OUTPUT

Every step answers one of the five questions — and the toast needed a decision and a repetition before the ball ever did. *This is an algorithm. Next question: how do you hand it to the literal assistant?*

Problem



Model



Compute



Analyze



Validate



Improve



Pseudocode: the toast, written for the assistant

THE TOAST, IN CAREFUL ENGLISH

1. Take one slice of bread
2. Set the dial to 3
3. Put the bread in
4. Press the lever down
5. Wait until it pops up
6. If too pale, press and
wait once more
7. Serve on a plate

THE SAME, AS PSEUDOCODE

→ GET one slice of bread
→ SET dial = 3
→ PUT bread in toaster
→ PRESS lever down
→ WAIT UNTIL lever pops up
→ IF toast is too pale
 PRESS and WAIT again
→ REPORT toast on a plate

WHAT CHANGED — AND WHY

PSEUDOCODE

The same algorithm, written English-like, one action per line, with a small set of doing-words: **GET, SET, COMPUTE, IF, WAIT UNTIL, REPORT.**

Still no programming language. No syntax to memorize — **precision is the only rule.**

Why bother? The doing-words map straight onto the five questions, so gaps show instantly — and this is the form you will hand to an AI, so the thinking stays yours.

Our four rules: one action per line · name every input and output · state units where they exist · any words work — there is no official syntax, only precision. Next: where this new notation sits.

Problem



Model



Compute



Analyze



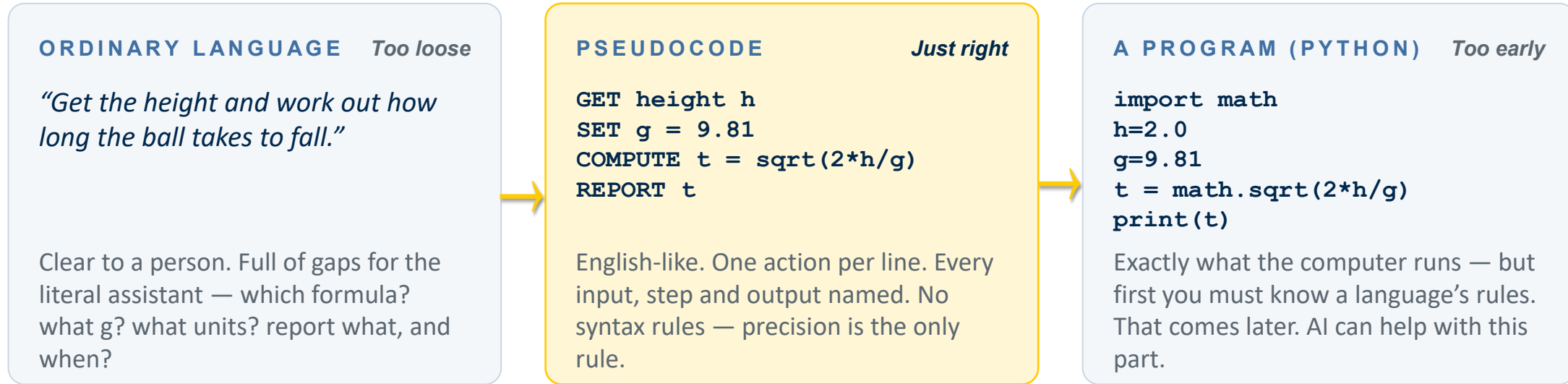
Validate



Improve

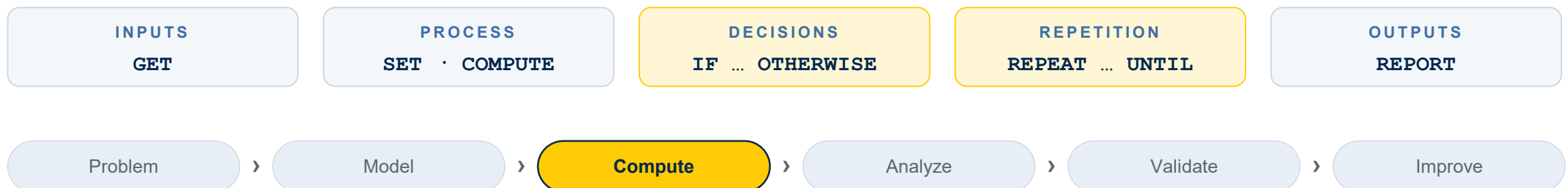


An algorithm lives in your head. How do you write it down?



Pseudocode is the middle step. It is plain English disciplined by the five questions you just met — every line answers one of them. We will think in pseudocode for the rest of this chapter, and only then let a language (or an AI) turn it into code.

THE FIVE QUESTIONS BECOME FIVE KINDS OF LINE





Now the fall time, in pseudocode

PSEUDOCODE

```
# PSEUDOCODE — fall time
GET height h           # meters
SET g = 9.81           # m/s^2
COMPUTE t = sqrt(2*h/g)
REPORT t               # seconds
```

ORDINARY LANGUAGE

“Get the height, then use the free-fall formula to figure out how long it takes, and tell me the time.”

Clear to a human. Full of gaps for a literal assistant: which formula? what value of g ? what units? report in what form? Exactly the gaps the toast had — and the same doing-words close them.



Pseudocode is an engineering document, not a programming trick.

It records your procedure precisely — before any language, and before any AI. It is how you will hand your thinking to AI without handing over the thinking itself.

Rules of our pseudocode: one action per line · name every input and output · state units · there is no official syntax — precision is the only requirement.

Problem



Model



Compute



Analyze



Validate



Improve



BEFORE THE NEXT EXAMPLE

What if a task does the same thing again and again?

TASKS LIKE THESE ARE EVERYWHERE

- ↺ Check the pressure in all four tires
- ↺ Test twelve samples, one by one
- ↺ Time the ball drop from five heights
- ↺ Draw a shape with four equal sides

THE QUESTION

Our pseudocode so far runs straight through — every line exactly once.

When one step must happen four times, do we write it four times? Forty times, if the task grows?

There ought to be a better word.

Let's find it the honest way — by feeling the pain first. *Next: a pen, four equal sides.*

Problem



Model



Compute



Analyze



Validate



Improve



Draw a square: sequence works, but notice the repetition

PSEUDOCODE · THREE-WORD PEN

PSEUDOCODE — draw a square

PEN DOWN

FORWARD 50

side 1

TURN LEFT 90

side 1

FORWARD 50

side 2

TURN LEFT 90

side 2

FORWARD 50

side 3

TURN LEFT 90

side 3

FORWARD 50

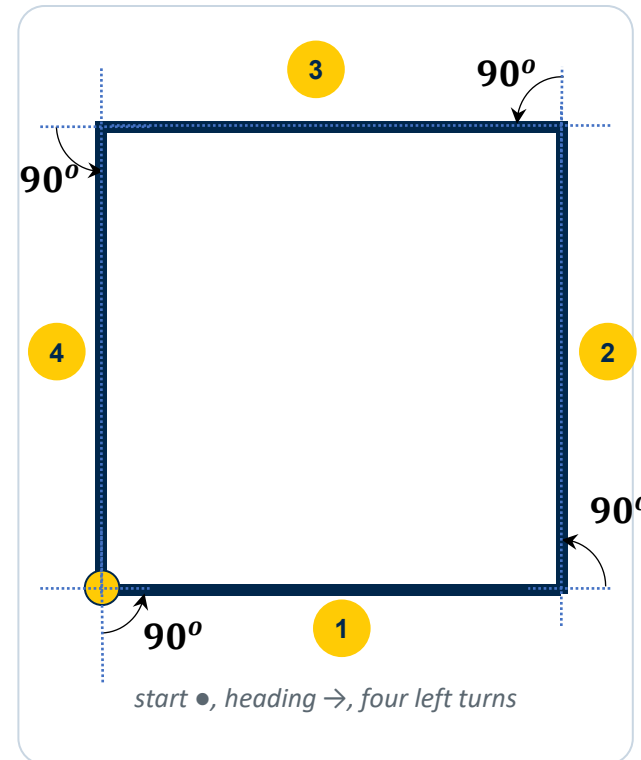
side 4

TURN LEFT 90

side 4

PEN UP

WHAT APPEARS ON THE PAPER



LOOK AT THE PSEUDOCODE AGAIN

Lines 3–10 are the same two lines, written four times.

Nothing is wrong with it — the literal assistant draws a perfect square. But a 12-sided shape would take 24 lines, and a 360-sided one 720. You would be typing what a machine should be repeating.

What we want to write instead:

REPEAT 4 TIMES

FORWARD 50

TURN LEFT 90

That word REPEAT — and its cousin IF — are what Part 2 is about.

Sequence is enough to describe a square. It is not enough to describe it well. *Real engineering procedures repeat and branch — the next part gives pseudocode the two words it still lacks.*

Problem



Model



Compute



Analyze



Validate



Improve



You can write an algorithm. Can you predict what it does?

WHAT YOU CAN DO ALREADY

WRITE an algorithm

Turn thinking into precise, ordered lines. You have now done it three times — the toast, the fall time, the square.

Writing is half the skill.

THE OTHER HALF

RUN one — in your head

Execute the lines exactly as the literal assistant would, and predict the result before anyone — or anything — runs them.

AI will help you write. Verifying what runs is not delegable.

Next: run the fall-time algorithm by hand, line by line, watching what the assistant remembers at every step. This is the habit every later chapter leans on.

Problem



Model



Compute



Analyze



Validate



Improve



What the literal assistant actually does with your lines

THE PSEUDOCODE · $h = 20 \text{ m}$

- 1 GET height h
- 2 SET $g = 9.81$
- 3 COMPUTE $t = \text{sqrt}(2 \cdot h / g)$
- 4 REPORT t

The assistant reads line 1, does it, forgets nothing, moves to line 2. Never two at once. Never out of order.

WHAT THE ASSISTANT REMEMBERS

after line	h	g	t
1	20	—	—
2	20	9.81	—
3	20	9.81	2.02
4	20	9.81	2.02

h , g , t are named boxes that hold a value — the values change as lines run; the names never do.
Line 4 sends 2.02 to the screen.

ORDER MATTERS

```
SET x = 3
SET x = x + 4
SET x = x * x
REPORT x → 49
```

Swap lines 2 and 3:

```
SET x = 3
SET x = x * x
SET x = x + 4
REPORT x → 13
```

Same four lines. Different answer. A procedure is a sequence, not a set.

Tracing by hand is how you verify an algorithm before anyone executes it. Pick a value you already know the answer for ($h = 20 \text{ m} \rightarrow$ about 2 s), walk the lines, watch the boxes. You will do exactly this to check what an AI hands you — the assistant that never asks what you meant is not going to catch a wrong line 3.

Problem



Model



Compute



Analyze



Validate



Improve



Write it. Trade it. Execute it literally.

1

WRITE (4 min)

Pick one everyday procedure — make instant coffee, parallel park, check tire pressure. Write pseudocode: one action per line, inputs and outputs named.

2

TRADE (1 min)

Swap with a neighbor. No explanations allowed. The paper must speak for itself.

3

EXECUTE (4 min)

Run your partner's procedure as the extremely literal assistant. Do exactly what is written — nothing more. Note every gap, ambiguity, and missing stop.

4

DEBRIEF (3 min)

Report the best failure you found. Where did the procedure assume common sense it never wrote down?

The point: a procedure that needs your explanation is not yet an algorithm.