

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,*

CHAPTER - 1

PART 3 · ≈ 45 MIN

From Model to Algorithm to Program

3

Code as an executable engineering model — plus the digits you should and shouldn't trust

Problem

›

Model

›

Compute

›

Analyze

›

Validate

›

Improve



Three representations of one idea



MODEL

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

Mathematics.

Compact, exact, human-readable
— but it doesn't run.



ALGORITHM

a recipe of steps

Plain language.

An unambiguous procedure
anyone (or anything) could follow.



PROGRAM

```
t = math.sqrt(2*h/g)
```

Executable.

The same idea, written so a
computer can carry it out.

Code is an executable representation of an engineering model — nothing more mysterious than that.



Name the pieces before writing any code



INPUTS

h drop height (m)

g gravitational acceleration (m/s²)

*Where do these numbers come from? Who measured *h*? Why 9.81?*



OUTPUT

t predicted fall time (s)

One number today. Later: tables, curves, optima, reports.



ASSUMPTIONS

- Released from rest
- Negligible air resistance
- Constant *g*

The program will NOT check these for you. They live in your head and your documentation.

Every computation this semester starts with this inventory — it takes two minutes and prevents most disasters.

Problem



Model



Compute



Analyze



Validate



Improve



The algorithm — a recipe with no ambiguity

- 1 Obtain the drop height h
- 2 Define gravitational acceleration g
- 3 Compute the quantity $2h / g$
- 4 Take the square root of that quantity
- 5 Report the fall time t

WHY BOTHER?

An algorithm is testable before it's typed.

You can walk it by hand, spot a missing step, or hand it to a teammate — or to an AI — and get back code in any language.

Notice: nothing here is Python. The thinking is language-free.



Each step becomes a line

	ALGORITHM STEP		PYTHON
1	Obtain the drop height	>	<code>h = 2.0</code>
2	Define gravitational acceleration	>	<code>g = 9.81</code>
3-4	Compute $2h/g$, take the square root	>	<code>t = math.sqrt(2*h/g)</code>
5	Report the fall time	>	<code>print(t)</code>

(Plus one housekeeping line — `import math` — to load the square-root tool.)

Problem



Model

**Compute**

Analyze



Validate



Improve

What is a computer program?

A computer program is a sequence of **instructions** that tells a computer how **to perform a task**.

Engineering Problem → Algorithm / Instructions → Program → Computer Executes → Results

Example: Python Program

```
import math
h = 2.0
g = 9.81
t = math.sqrt(2*h/g)
print(t)
```

The same algorithm can be expressed in different programming languages — **Python**, **MATLAB**, **VBA/Excel**, C++, Java, and others.

Analogy: Recipe for cooking omelet



Get the eggs, spices, pan, whisker
Crack the eggs in the pan.
Add salt then stir
Add milk and spices on the top
Cook at gentle heat
Serve

The same recipe can be written in different languages **English**, French, German, Spanish, Arabic, etc. to cook the omelet.



Every line has an engineering meaning

Command (key word)

Library Module

is for your Comment (any text after that is not executed) skipped

```
import math
```

```
h = 2.0 # height (m)
```

```
g = 9.81 # m/s^2
```

```
t = math.sqrt(2*h/g)
```

```
print(t)
```

```
0.6385508568141011
```

Command
or program
lines

`import math` Load toolbox of math operations ($\sqrt{}$, π , trig, ...)

`h = 2.0` Store the measured physical parameter: height

`g = 9.81` Define a parameter of our physical model

`t = math.sqrt(2*h/g)`
Evaluate the mathematical model computationally

`print(t)` Report or Display the engineering result

Execute (RUN) the program. (Click play button)

Output displayed after running the program

Do not focus on “Python syntax” — simply programmed executable version of $t = \sqrt{2h/g}$.



Seven ideas — you already used them all

Variables

h, g, t — named storage for quantities

Functions

`math.sqrt(...)` — a named, reusable operation

Assignment

`=` means “store this value under this name”

Comments

`#` text the computer ignores; humans need it

Numeric values

2.0 and 9.81 — numbers with decimals

Output

`print(...)` — show a result to the human

Arithmetic

`*` `/` `+` `-` work like your calculator

Not this week: loops · conditionals · OOP · NumPy · plotting. They come when a problem needs them.

Rusty or brand new? This list is exactly what to review with your AI tutor tonight.

Problem



Model



Compute



Analyze



Validate



Improve



Our environment: Python + VS Code + Jupyter

In [1]:

```
import math
h = 2.0
g = 9.81
t = math.sqrt(2*h/g)
print(t)
```

Out: 0.6385508568141011

Run cell · edit · run again — that's the whole loop

A notebook mixes runnable code cells with text, math, and plots — ideal for engineering investigations you can re-run and share.

BY THE END OF LAB, YOU CAN:

- Open VS Code + a notebook
- Execute a cell
- Change a value (try $h = 4.0$)
- Re-run and watch the output change

That's it. Setup instructions are posted — install before lab so lab time goes to engineering.



DISCUSS



The program ran with no errors.

Does that mean it is correct?

What, exactly, does “it ran” prove? Make a list of what it does NOT prove.



The computer is very confident

Measured height (tape measure):

$$h = 2.03 \text{ m}$$

Python reports:

$$t = 0.643322162442 \text{ s}$$

Twelve digits.

Does having twelve digits make this answer accurate?

Which digits would you sign your name to?

Hint: the tape measure knows h to about the nearest centimeter...



First exposure — the full treatment comes later



Accuracy

How close a result is to the true value.



Precision

How finely a result is expressed or how repeatable it is — possibly precisely wrong.



Significant figures

The digits that actually carry information. Report these; drop the noise.



Measurement uncertainty

Every measurement has a $\pm$. It limits how many digits any downstream answer deserves.

Computers can give many digits. Engineering measurements do not necessarily justify those digits.