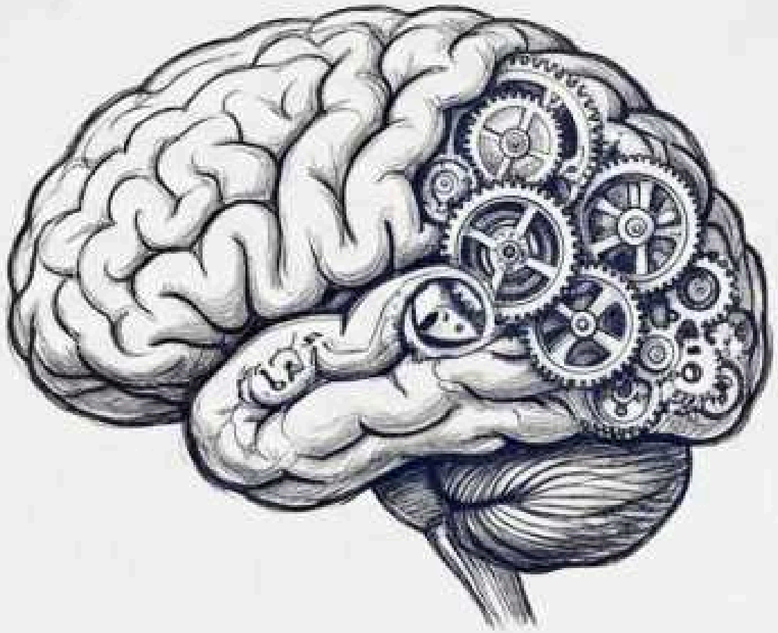
[¶3.1]

Python Basics: Values, Variables, and Types.



¶ 3.2

Programming is a skill of the head...



...and a skill of the hands.



A few minutes at a keyboard will teach you more than any amount of listening. Experiment as we go.

¶ 3.3

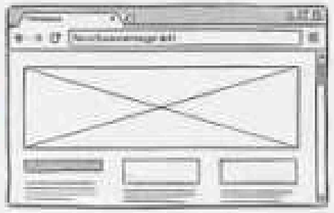
The namesake
(not a snake)



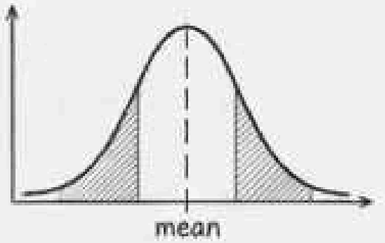
1991:
Guido van Rossum



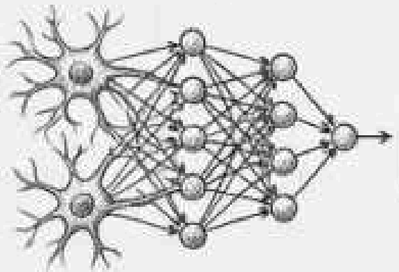
Science



Web
Development



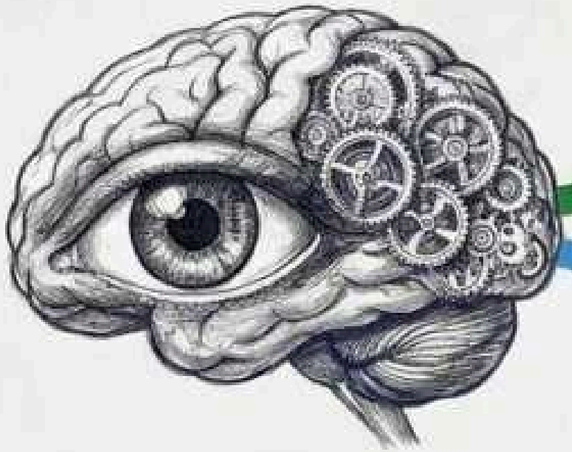
Data
Analysis



Artificial
Intelligence

Designed for readability. Ideas learned here transfer to hundreds of other languages.

¶3.4



Interactive Prompt

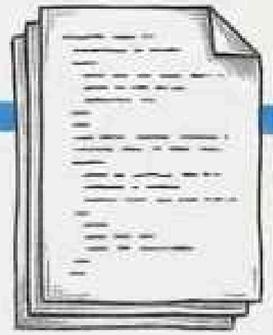
Instant result

Type a line,
see the result at once
(for experimenting).

THE INTERPRETER
(Reads & carries out instructions)

Final result

File Mode



Save many lines,
run all together.

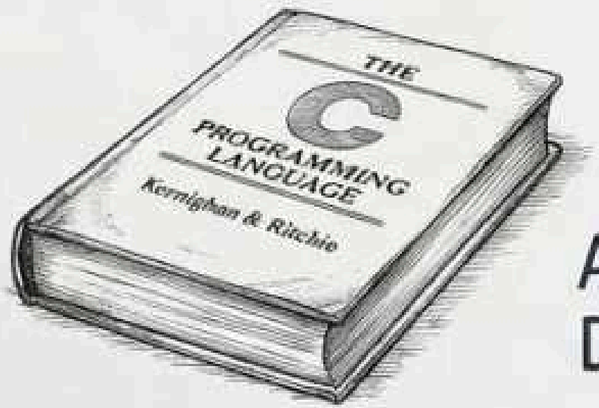
¶ 3.5

Function:
A named action.

```
print("Hello, world")
```

Container for what the function works on.

The target value.



A tradition tracing back to Brian Kernighan and Dennis Ritchie's C language.

¶3.6

Integers	Floats
(Whole numbers)	(Decimals)
42, -7	3.14, 0.0
Strings	**Booleans**
(Text in quotes)	(True/False)
"Python", "123"	True, False

Math: 3 + 4 = 7 Strings: "3" + "4" = "34"

[¶ 3.7]



\+

\-

*

** (Power)

/ (Division → always gives a float: 7 / 2 = 3.5)

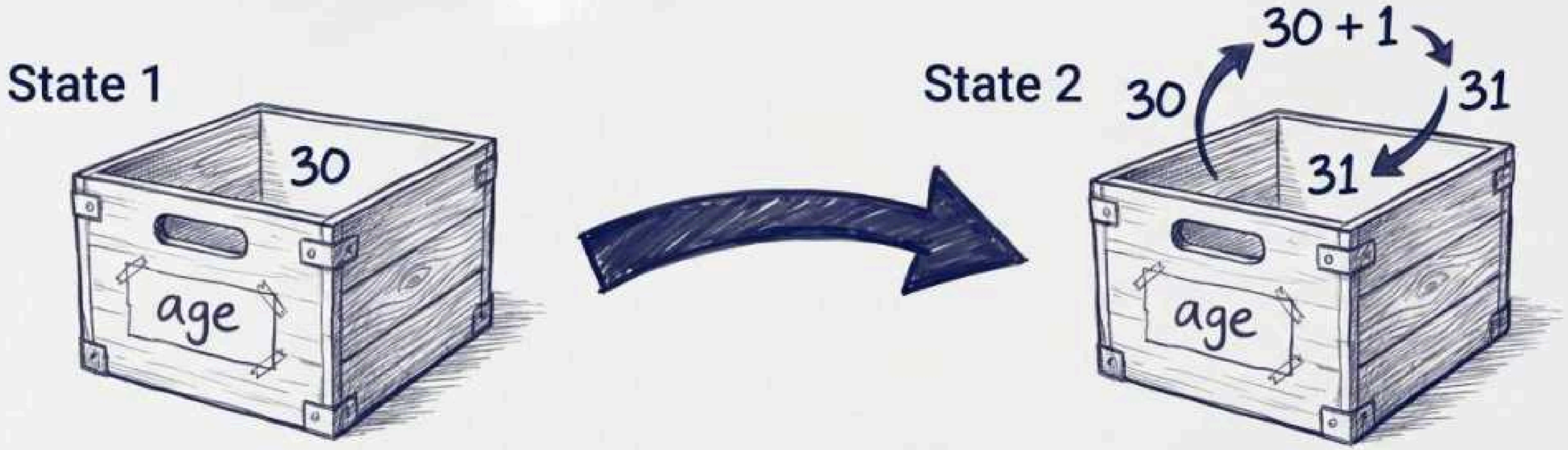
// (Double Slash)
: Whole-number quotient (7 // 2 = 3)

% (Percent / Modulo)
: The remainder (7 % 2 = 1)

Parentheses and standard order of operations apply, just as in algebra.

[¶3.8]

Read "=" as SET TO, not "Equals".




```
age = 30
```

```
age = age + 1
```

Take the old value, add one, store the result.

[¶3.9]

Rules: Letters, digits, underscores.
Cannot start with a digit. Case-sensitive.





`total_price`

Convention: lowercase
joined by underscores.
Acts as free documentation.

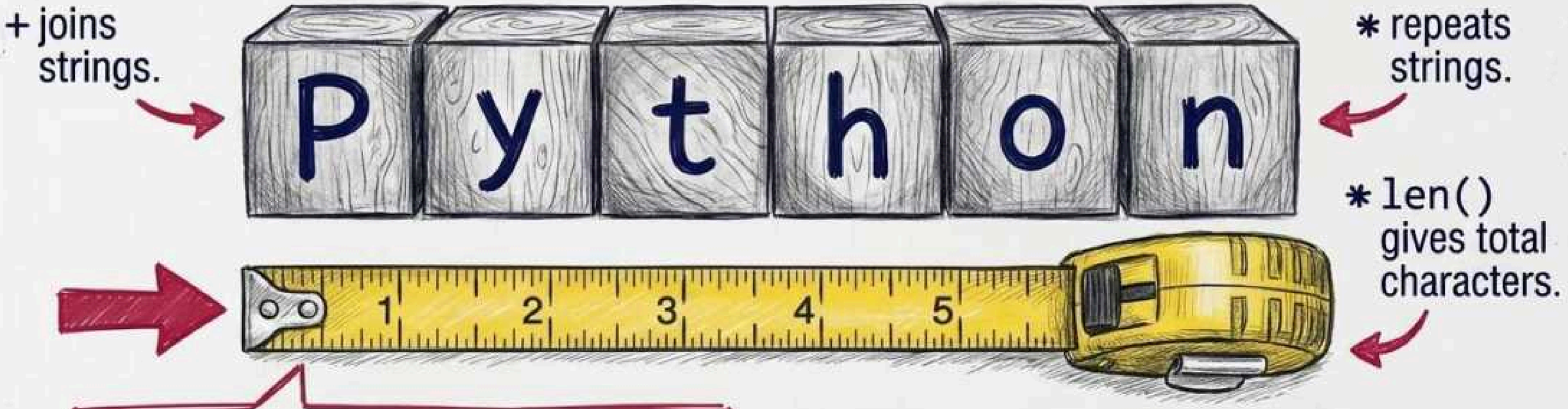
`1st_price` (Starts with digit)

`TotalPrice` (Violates convention)

`q` (Tells the reader nothing)

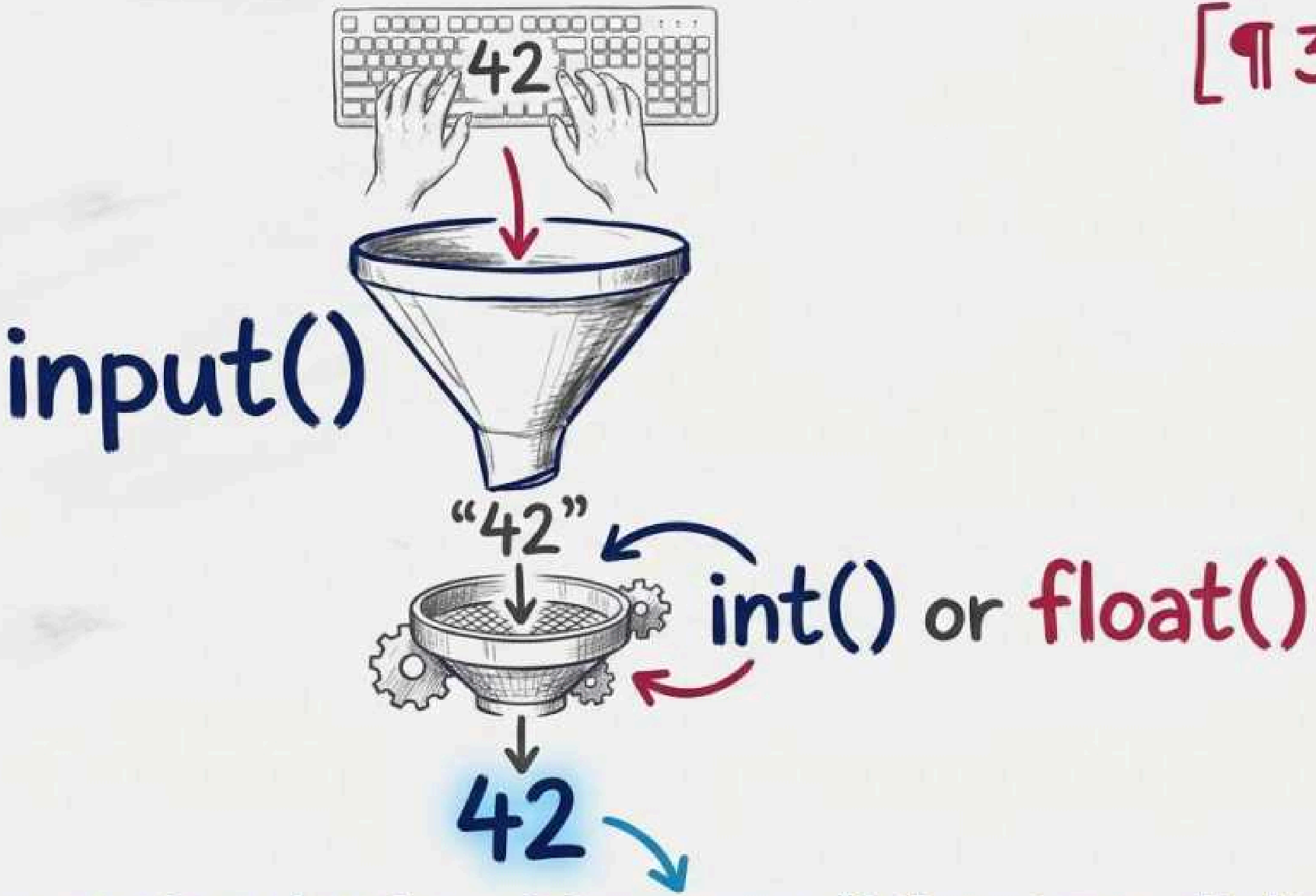
Choose names that say what the value means.
"total_price" costs nothing; "q" tells the next reader nothing.

[¶ 3.10]



Zero-Indexing: Position is the distance from the start. The first character is zero steps away.

[¶ 3.11]



input() always gives back a string, even if they type digits.
You must convert it to use it as a number.

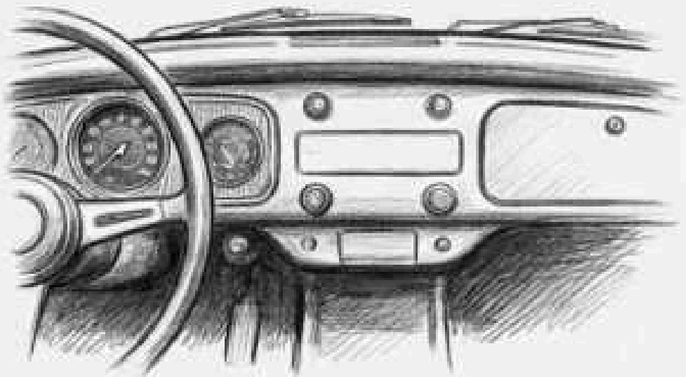
[¶ 3.12]



f-strings: Build text from values with no awkward joining.
Formatted output appears in almost every real program.

[¶ 3.13]

Syntax Error



Grammar is wrong.
Program will not start.

Runtime Error



Starts, but crashes during execution (e.g., adding a string to a number).

Logic Error



Runs perfectly, gives the wrong answer.
The hardest to find.

Errors are a normal part of programming.
Learning to read them is a core skill.

[¶ 3.14]

Python ignores everything after the #. This line is a comment for humans, not the computer.

```
# Adjusting for timezone offset
time = time + 2
```

This line is executed by Python.

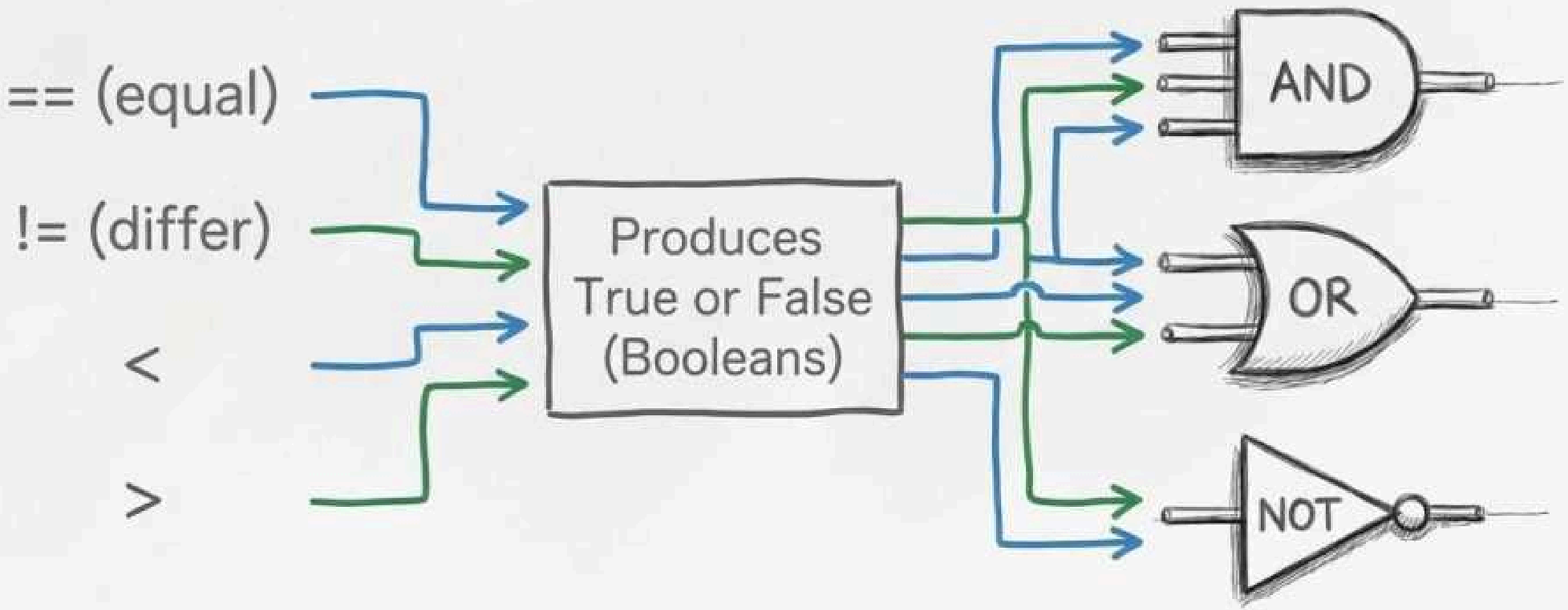


A good comment explains the WHY behind the code, providing context like this calendar entry.

Python ignores everything after the #. Good comments explain WHY the code does something, not WHAT it does. Code is read far more often than it is written—clarity is a kindness to your future self.

[¶ 3.15]

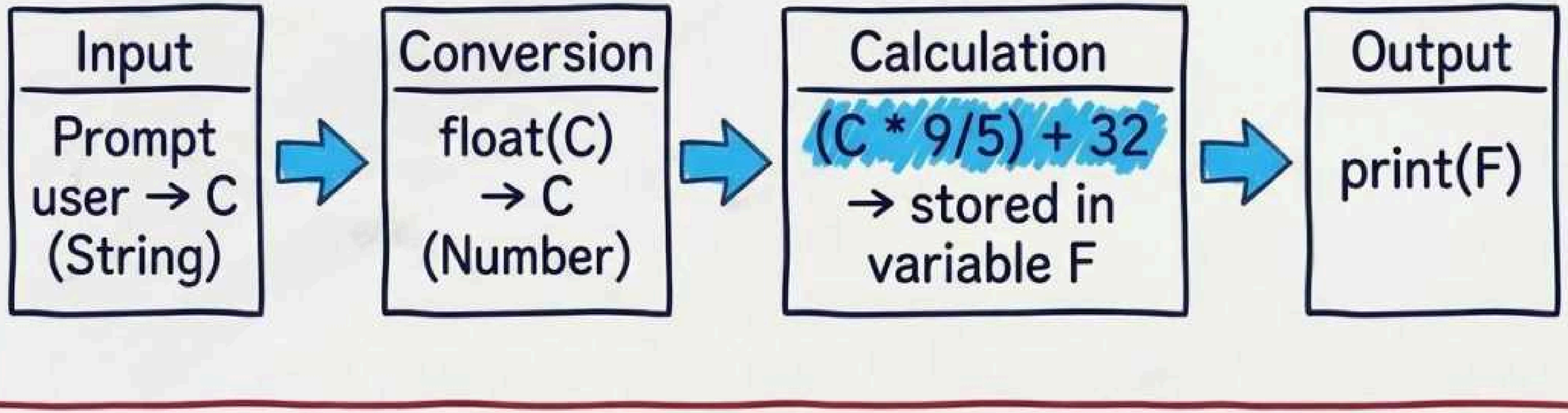
Logic Gate Equivalence



Comparisons produce booleans. 'and', 'or', and 'not' combine them exactly like the hardware logic gates of Lecture 2.

[¶ 3.16]

System Architecture Flowchart: Temperature Converter



Bringing it all together. Test cases to verify logic: 0°C → 32°F, and 100°C → 212°F.

[¶ 3.17]

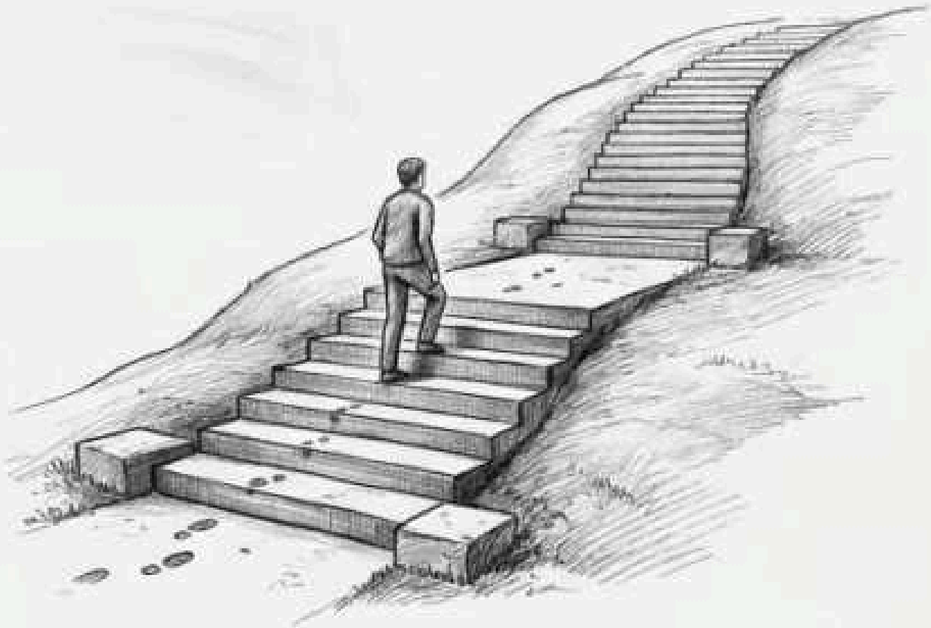
Build programs in small steps, and run them often.

The Cliff



Writing 20 lines before running.
Facing a dozen errors at once.

The Stairs



Writing 2–3 lines, running often.
Faults are isolated to the last few lines.

[¶ 3.18]





Which variables are needed?

Which values are strings and which are numbers?

Where must type conversion happen?







The Challenge: Write the steps of a program that asks for the price of an item and a tax rate, then prints the total cost.

[¶ 3.19]



Next in Lecture 4: We teach programs to make decisions and to repeat themselves.