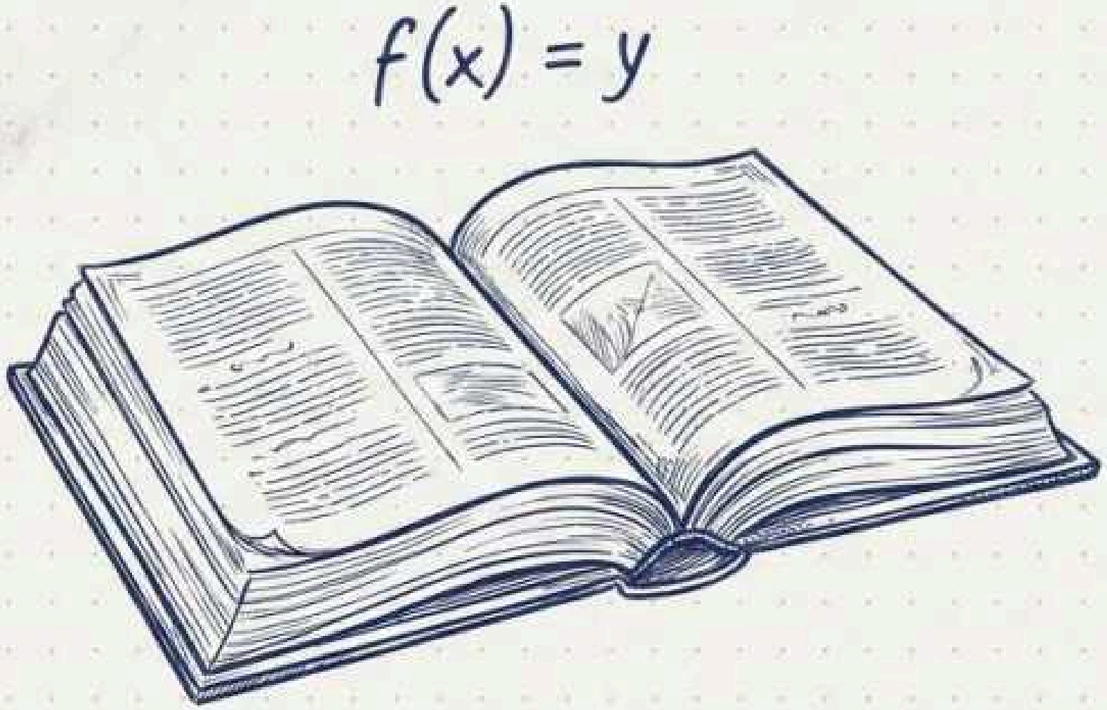
¶ 5.1



Lecture 5

Functions, Lists, and Organizing Code

- Functions & Scope
- Collections (Lists, Sets, Dictionaries)
- Files, Debugging & Testing

The Two Problems of Growth

¶ 5.2

Repetition

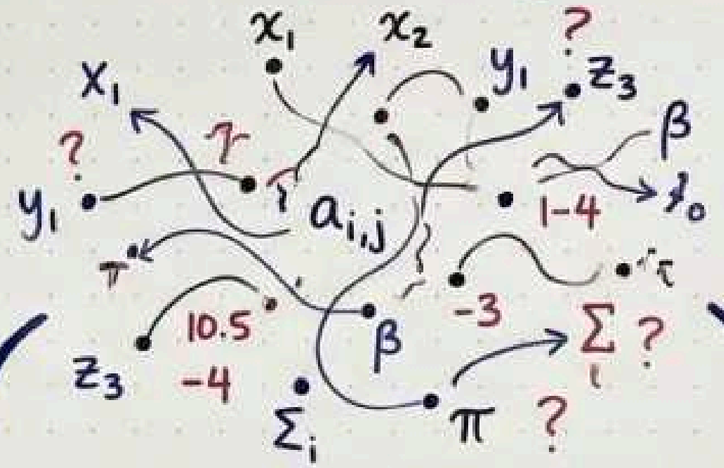
```
do_something() {
  process_data();
  calculate_x()
  ...
}

do_something() {
  process_data();
  calculate_x()
}

do_something() {
  process_data();
  calculate_x()
  ...
}
```

Functions

Complexity



Collections

Functions solve repeated code. Collections solve varied data.

The Anatomy of Abstraction

¶ 5.3

Definition

Name

Parameter (Input)

```
def square(x):
    return x * x
```

output

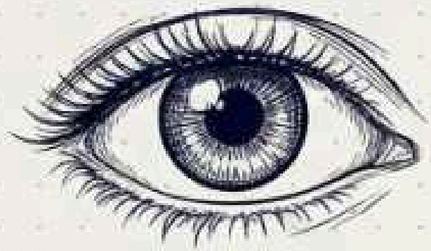
$f(5) = 25$

Use the name, forget the details.

Diverging Paths: Print vs. Return

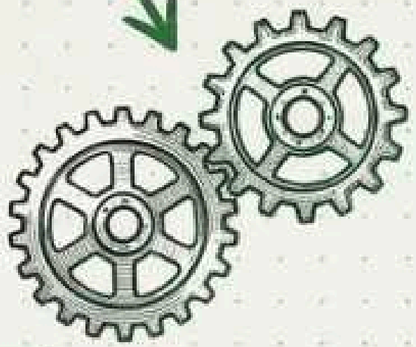
¶ 5.4

`my_function()`



The Screen

Print shows data to humans.
Cannot be used in calculations.



The Program

Return hands a value back to the calling program.

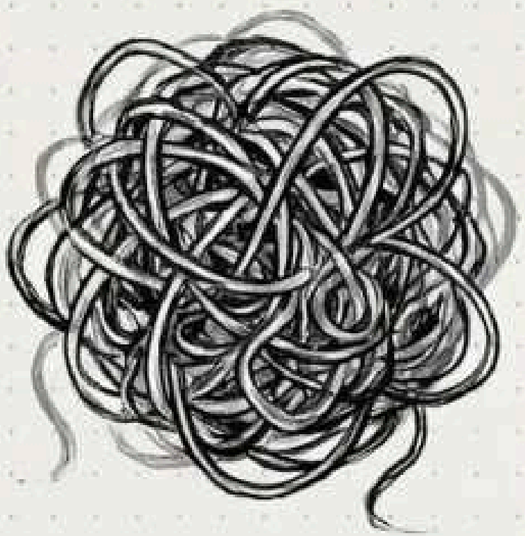


A function with no return statement quietly gives back None—a common cause of puzzling bugs.

Single Responsibility

¶ 5.5

Before



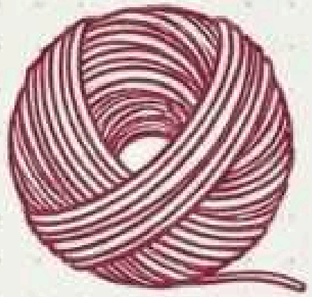
After



C → F



F → C

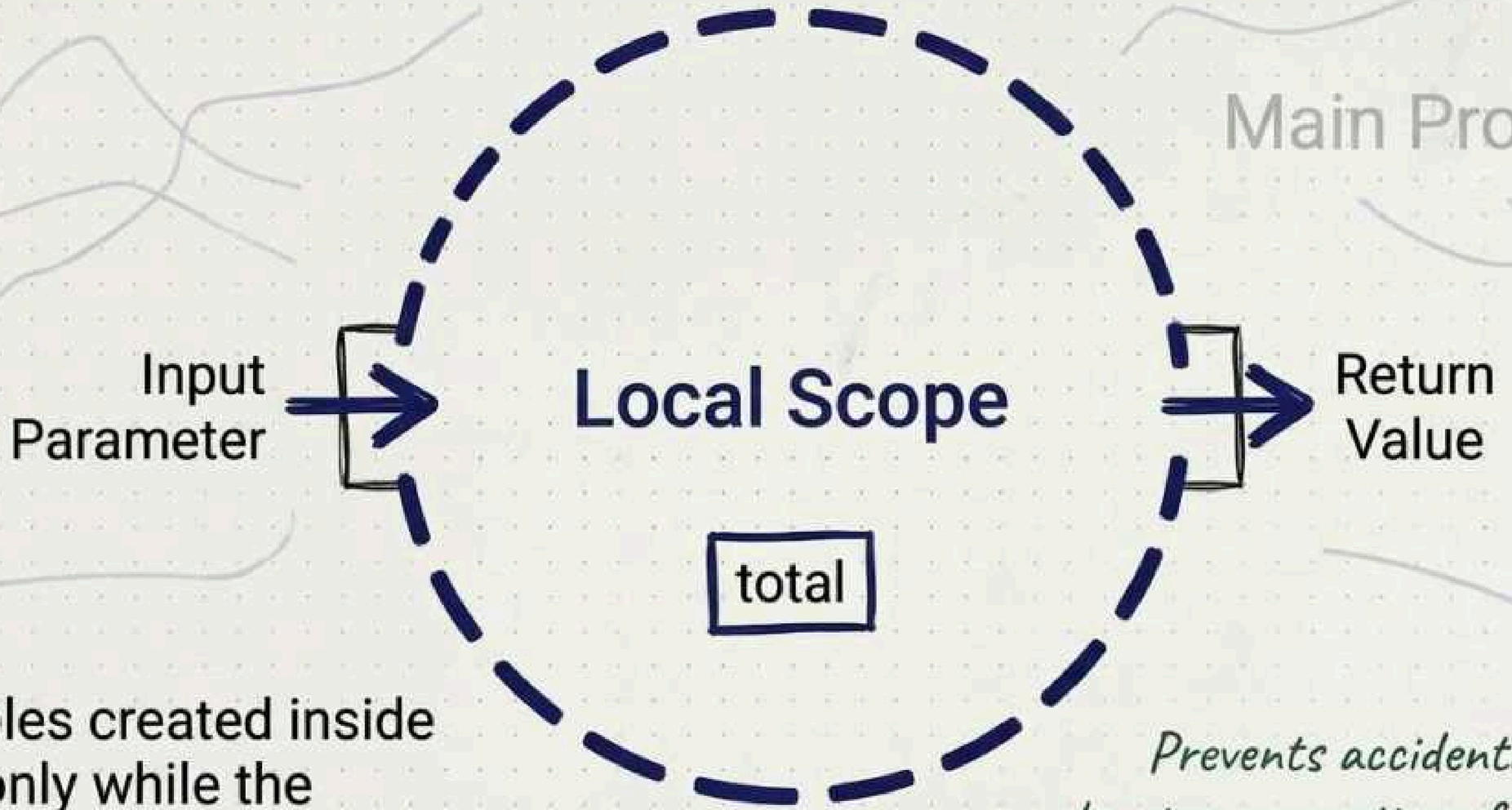


Print Table

- Good functions are small and do one job.
- Easily tested separately.
- Reusable in later programs.
- Understood at a glance.

The Boundaries of Local Scope

¶ 5.6

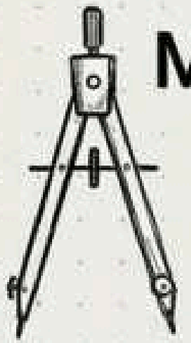


Variables created inside exist only while the function runs.

Prevents accidental clashes, keeping connections few and clear.

The Standard Library

¶ 5.7



Modules

math

$\sqrt{(x)}$, $\sin(\theta)$



random



Built-ins

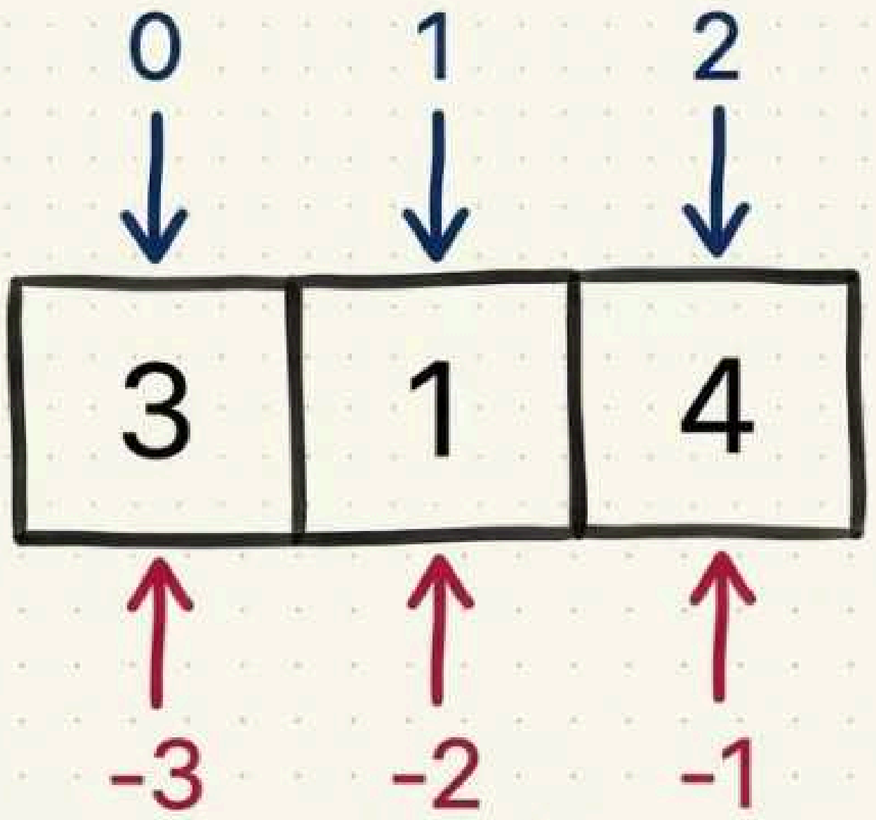
`len`, `max`, `min`, `sum`, `abs`, `round`

Before you write a function, ask whether someone has already written and tested it.

Lists (Ordered and Mutable)

¶ 5.8

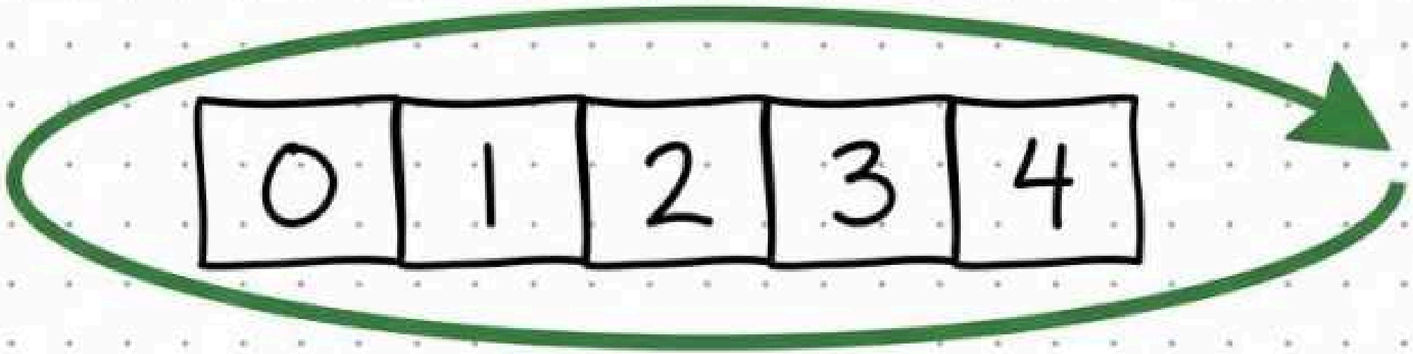
- An ordered collection, reached by position.
- Mutable: Can be changed after creation.



Actions:
append (add)
remove (delete)
slice (extract)

Lists and Loops Belong Together

A **for** loop visits every item in sequence.



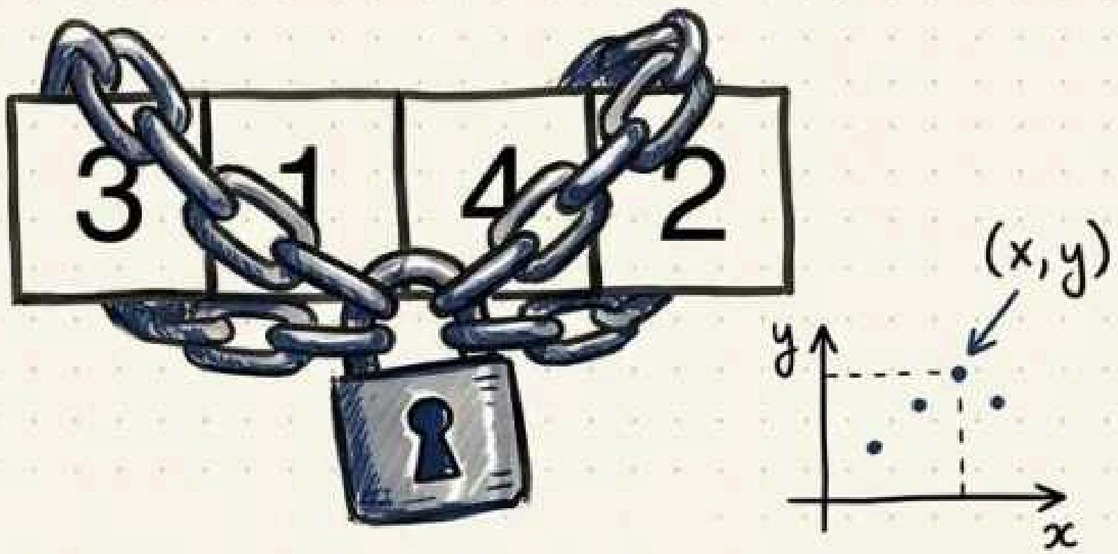
$$\mu = \frac{\sum x_i}{N} \Longrightarrow \text{sum(list) / len(list)}$$

List Comprehension:

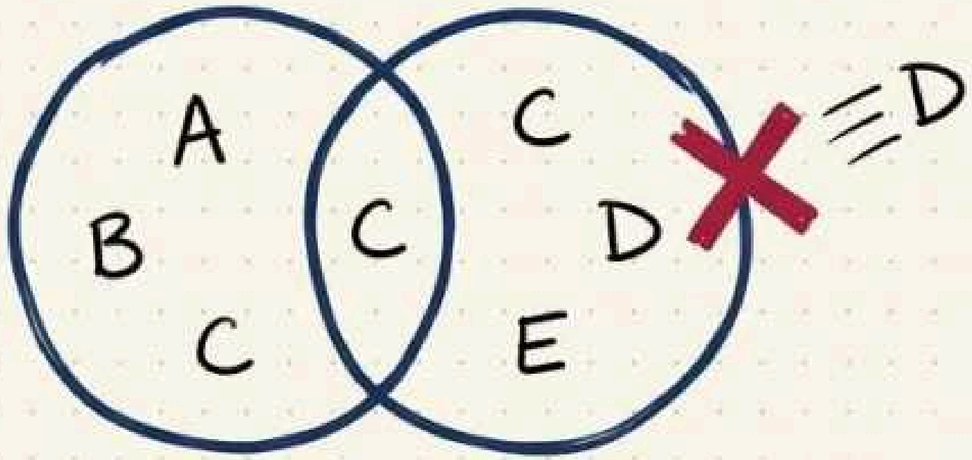
[x*x for x in range(5)] → [0, 1, 4, 9, 16]

Tuples and Sets

¶ 5.10



Tuple: Like a list, but cannot be changed (immutable). Suits fixed groups.

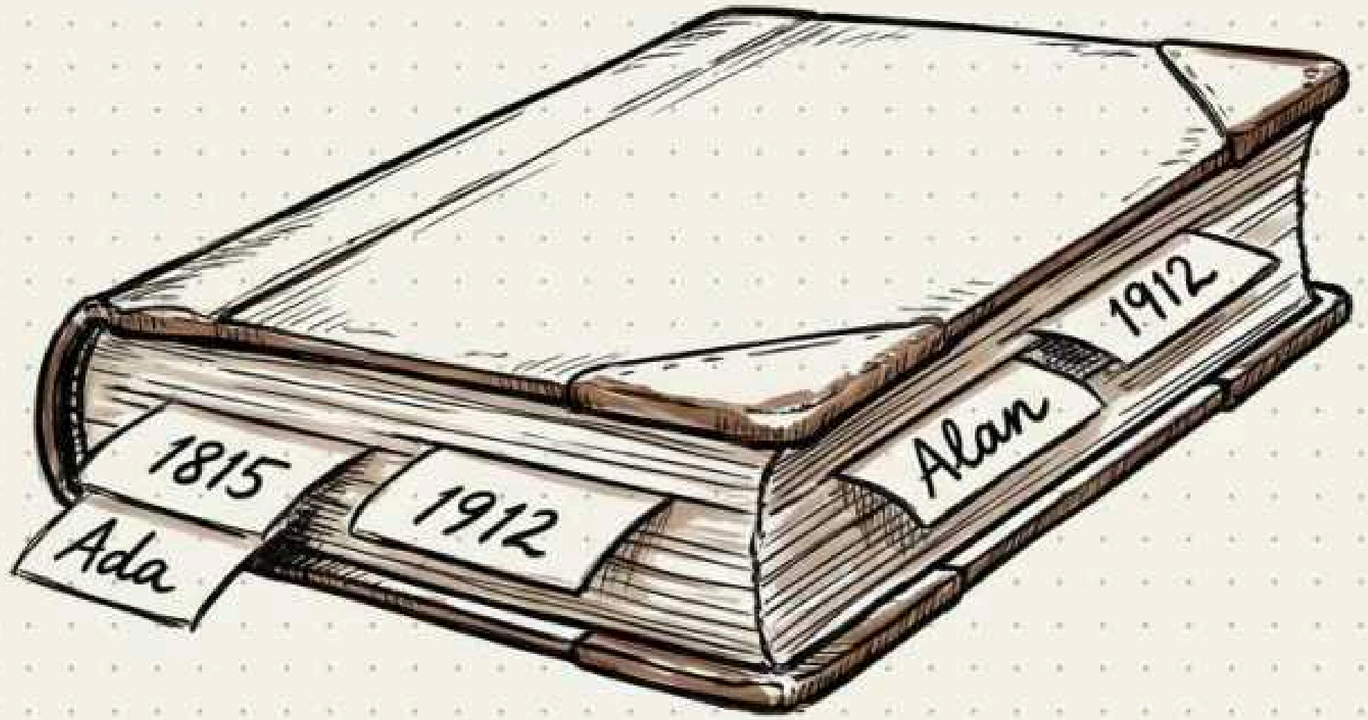


Set: Holds each item at most once. Extremely fast for answering: “Is this item in the collection?” (Removes duplicates).

¶ 5.11

Dictionaries: Key-Value Pairs

```
{"Ada": 1815, "Alan": 1912}
```

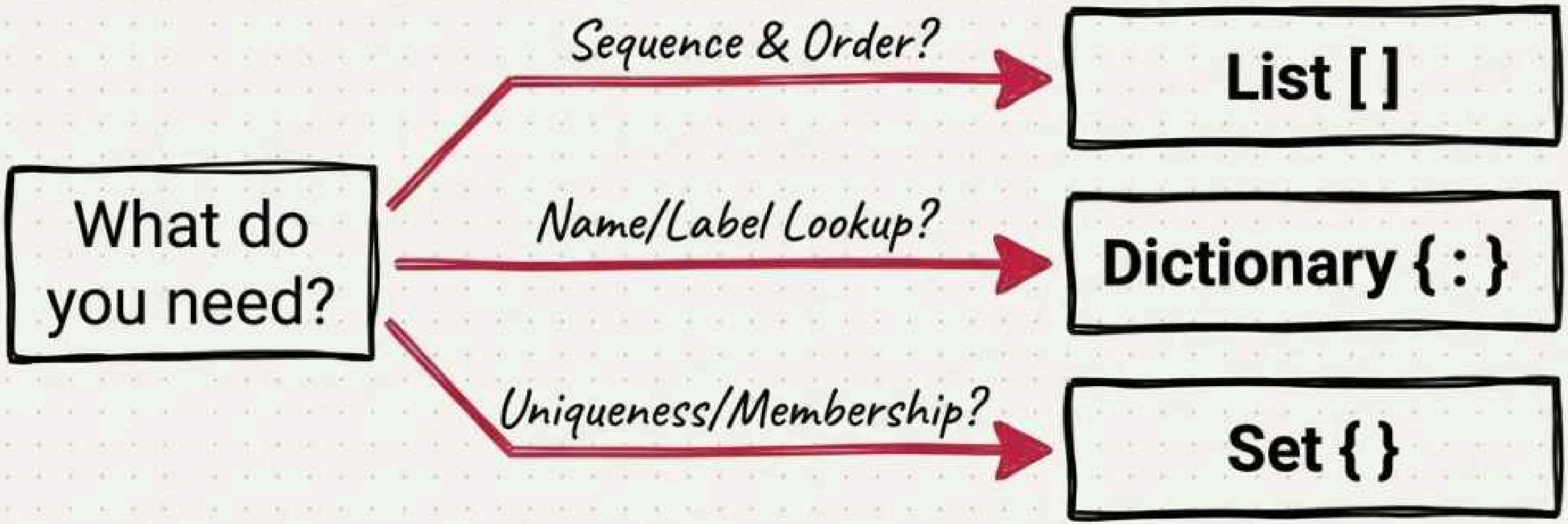


- Think of a phone book: look up a name, get a number.

- Finds values **almost instantly**, even with millions of entries.

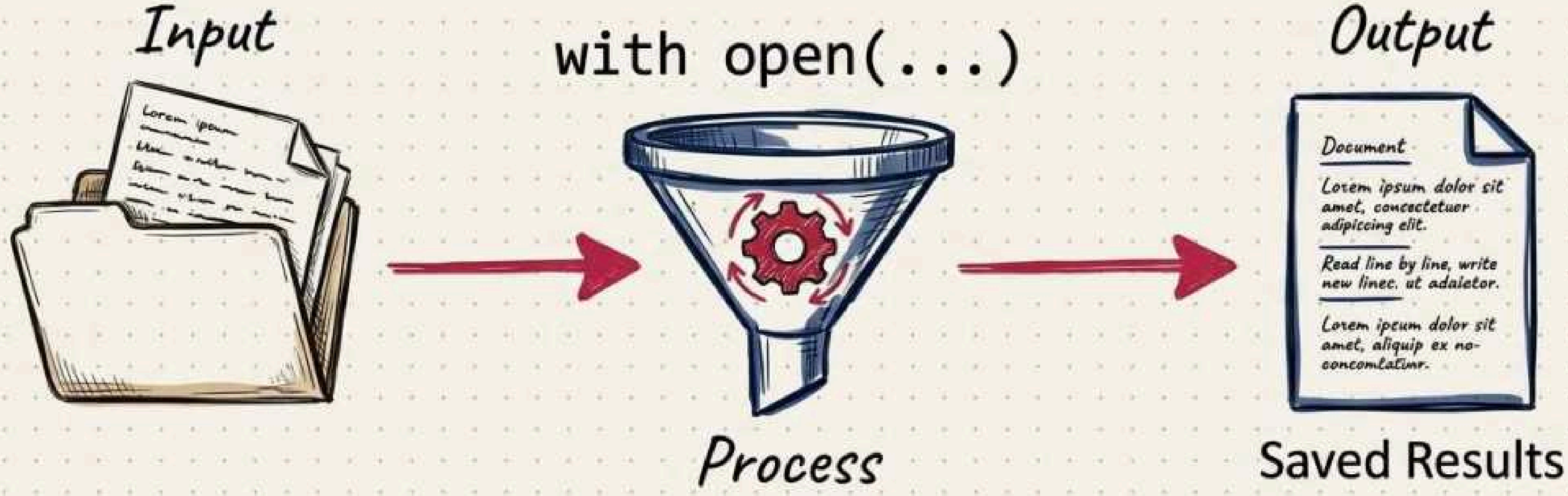
- Ideal for **counting occurrences**.

Choosing the Right Structure



Choosing the right structure is a design decision. The right choice makes code shorter, faster, and clearer. The wrong one makes everything awkward.

Files and Persistence

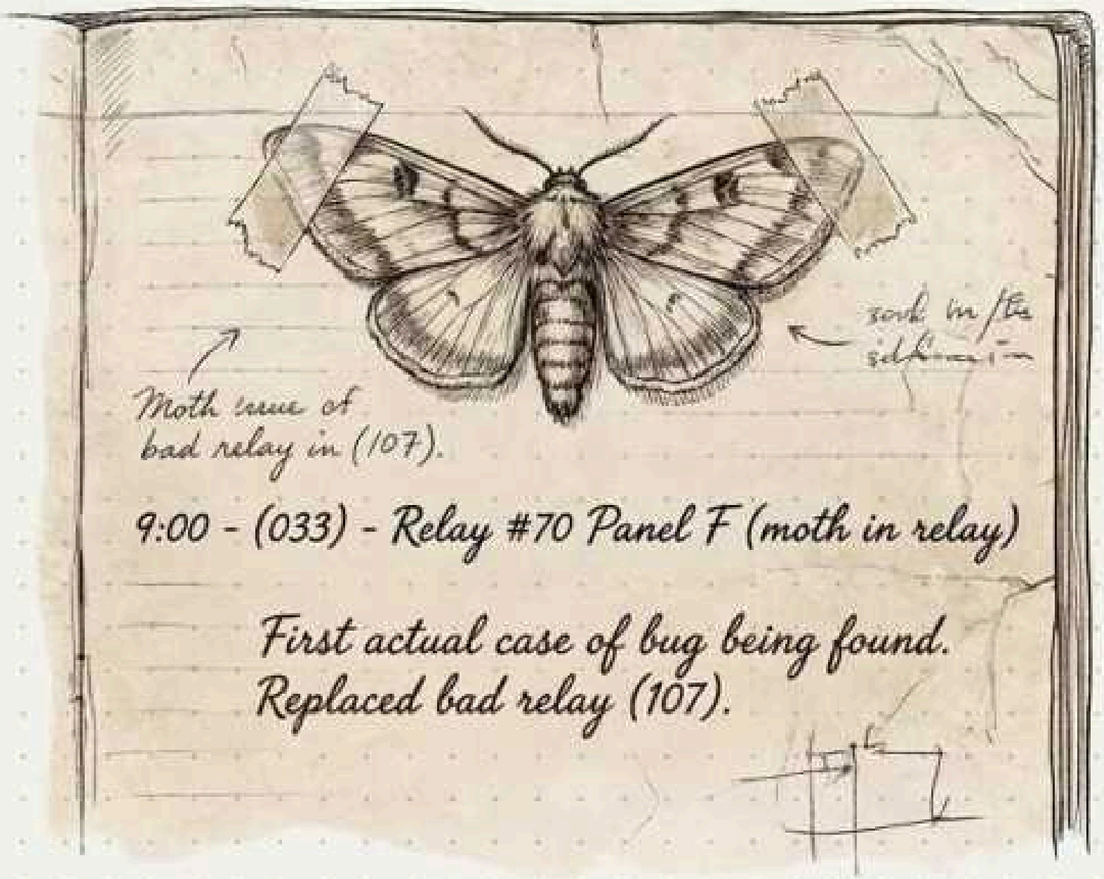


- Keep data after the program stops.
- Read line by line, write new lines.
- The **'with'** block acts as a **safety net**, automatically closing the file when finished.

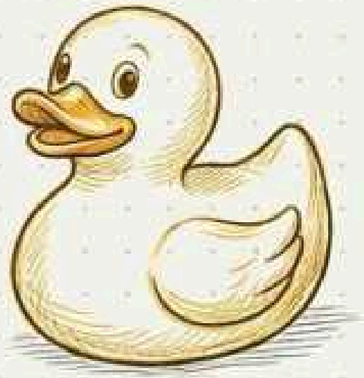
The first step from toy programs to useful tools.

Timeless Debugging Strategies

Historical Context



Actionable Checklist



1. Read the error message.
2. Reproduce the problem.
3. Print variable values.
4. Test small pieces.
5. Explain the code aloud to a rubber duck.

Testing with Assertions

- Decide inputs and outputs before you write the function.

- Tests protect you later when new changes break code that used to work.

Pass

```
assert square(3) == 9
```

(Passes silently)



Fail

```
assert square(3) == 10
```

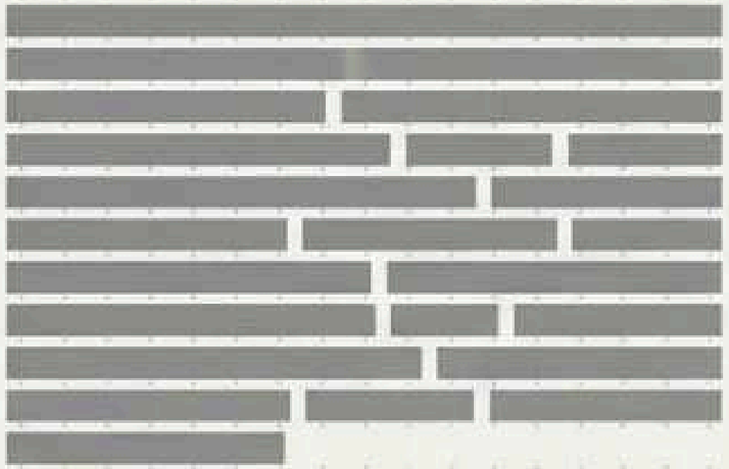
(Wrong answer shouts)



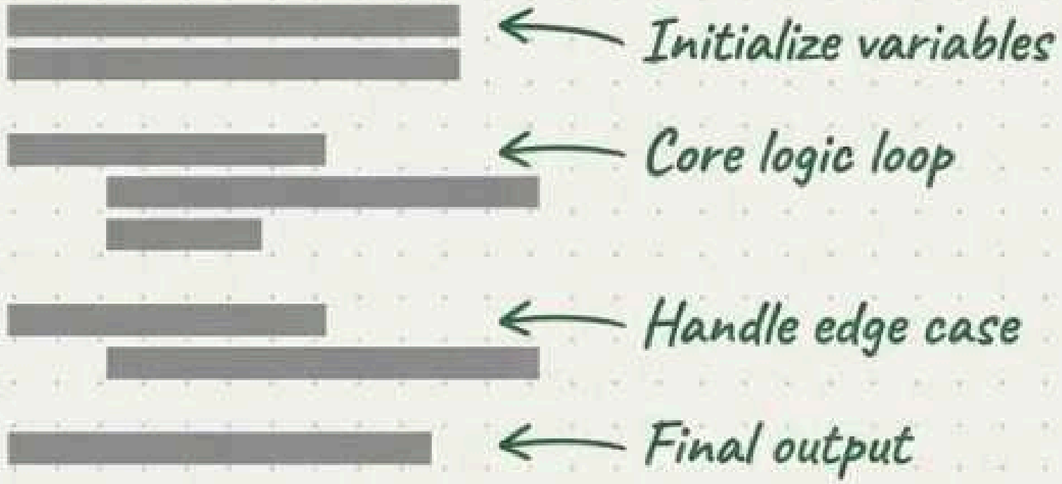
¶ 5.16

Style and Maintenance

Bad Code



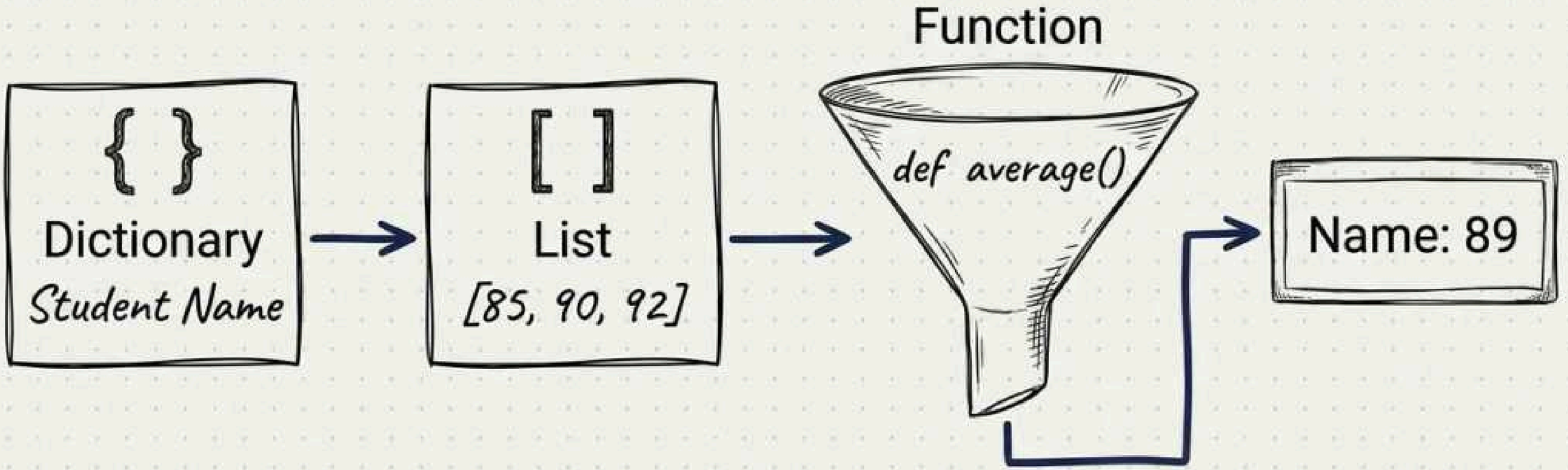
Good Code



- Use descriptive names.
- Keep functions short.
- DRY: Don’t Repeat Yourself (if it appears twice, make it a function).

“The next reader of your code is usually you, a few months from now.”

Synthesis: The Grade Book



Three core ideas combine to make a useful tool in about 10 lines of code.

The Challenge: A Small Library

¶ 5.18



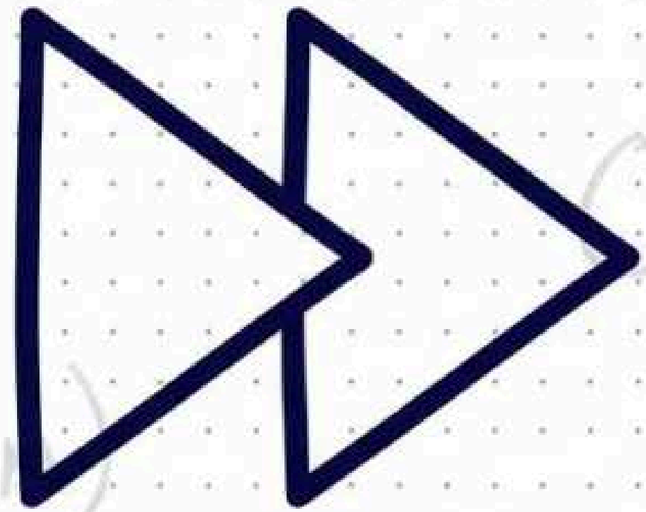
You must track library books (Titles, Authors, Loan Status).

- Which structures would you choose, and why?
- Describe one function you would write (inputs and results).
- Name one test you would run.

Looking Ahead

¶ 5.19

Next up: Lecture 6



How fast are algorithms, and where does computer science go from here?