Day 4

What is the walrus operator?

The walrus operator `:=` is an **assignment expression** introduced in Python 3.8. It lets you **assign a value to a variable as part of an expression**, instead of having to do the assignment on a separate line.

Simple contrast:

```
# before (no walrus)
value = f()
if value:
    use(value)

# with walrus
if (value := f()):
    use(value)
```

Why it's relevant

- **Reduces repetition**: avoids calling the same function twice or repeating an expression.
- **Keeps code concise**: assigns and tests in a single expression (especially in `if`, `while`, comprehensions, generator expressions).
- **Can improve performance** slightly by avoiding duplicate function calls.
- **Makes some patterns more natural**, like reading chunks from a file or grabbing regex match objects and using them immediately.

Where you can use it

Common places where it shines:

- `if` and `while` statements
- List/dict/set comprehensions and generator expressions

- Anywhere an expression is allowed (but *not* in plain assignment statements; it's an expression, not a statement)

Practical examples

1) Read chunks in a loop

```
with open("large.bin", "rb") as f:
    while (chunk := f.read(4096)):
        process(chunk)
```

No need to write `chunk = f.read(4096)` on a separate line.

2) Regex matching

```
import re
if (m := re.match(r"(\w+)=(\d+)", s)):
    key, num = m.group(1), int(m.group(2))
```

You both get the match object and test it in one line.

3) Avoid repeated work

```
def expensive(x): ...
if (res := expensive(x)) > 0:
    do_something(res)
```

Call `expensive()` once and reuse its result.

4) Use in comprehensions / generator expressions

```
# build list of results where result > 0, avoiding double call
results = [res for x in inputs if (res := f(x)) > 0]
```

(This is concise but be mindful of readability — see caveats.)

5) In generator expressions

```
total = sum((n := compute(i)) for i in range(10))
# though here the assignment is usually less useful; use case-specific
```

Gotchas & best practices

- **Requires Python 3.8+**. Code using `:=` won't run on older Python versions.
- **Readability**: don't sacrifice clarity for terseness. If an expression becomes hard to parse, split it into two lines.
- **Precedence & grouping**: the walrus is an expression — when combined with other operators, prefer parentheses to avoid surprises:

```
# safer
if ((m := f(x)) and m.something()):
    ...
```

- **Scoping subtleties**: assignment expressions assign into the current scope. In some comprehension contexts you should be careful about which variables become visible where; if unsure, test or assign before the comprehension.
- **Don't overuse**: it's great for reducing duplication, but avoid clever one-liners that become cryptic.

When not to use it

- If doing so makes the line hard to read (prefer explicit assignment).
- If you need to support Python versions < 3.8.
- If the assignment is complex (multiple steps) — use separate statements.