Week 1 - Lesson 5 Summary

Topic: Type hints + clean code habits (refactoring your CLI into small reusable functions).

Learning goals

- Read and write common Python type hints (without overthinking).
- Use small functions to keep your main loop clean and understandable.
- Use a helper that returns **int or None** for safe input validation (reusable code).
- Understand the basic purpose of **if __name__ == "__main__":**.

A) Type hints you will actually use

Type hints are **expectations** (mainly for readability and editor/tooling help). Python usually does not enforce them at runtime unless you add checks.

Common patterns:

```
def add(a: int, b: int) -> int:
    return a + b

def clean_text(s: str) -> str:
    return s.strip()

def save_items(items: list[str]) -> None:
    ...

def find_first(items: list[str], target: str) -> int | None:
    for i, x in enumerate(items):
        if x == target:
            return i
    return None
```

Key takeaway from our chat: in **enumerate(items)**, **i** is the index (int) and **x** is the value (same type as the list elements).

Type hints vs runtime behavior

If you annotate something as **str** but pass an **int**, Python will not automatically throw an error. It only breaks when you call a string-only method on an int (e.g., **.strip()**). If you want to enforce types, you can use **isinstance** checks and raise **TypeError**.

```
def must_be_str(x: str) -> str:
    if not isinstance(x, str):
        raise TypeError("x must be a string")
    return x.strip()
```

B) Clean code habits for your CLI

- Keep the main loop small: show menu -> read choice -> call a function.
- One job per function (add, list, remove, save, load).

- Use a helper for repeated validation (e.g., reading an index).
- Prefer clear names over clever one-liners.

Reusable helper: read_index

Instead of printing inside the helper, return **None** when invalid and let the caller decide what to display. This makes the helper reusable in different places.

```
def read_index(prompt: str, max_exclusive: int) -> int | None:
    s = input(prompt).strip()
    if not s.isdigit():
        return None
    idx = int(s)
    if idx < 0 or idx >= max_exclusive:
        return None
    return idx
```

Important: **return** ends the function immediately and sends a value back. Returning **None** is a clean way to signal "invalid".

C) The __main__ guard

This pattern runs your program only when the file is executed directly (e.g., **py main.py**), not when it is imported by another file.

```
def main() -> None:
    ...

if __name__ == "__main__":
    main()
```

Mini practice (with answers)

1) What is the return type of **read_index**?
Answer: **int | None**

2) If max_exclusive = 2, what does it return for different inputs?

User input	Return value	Reason
"1"	1	Valid digit and in range [0, 1]
"5"	None	Out of range
"abc"	None	Not digits

Quick rules to remember

- Type hints describe what you expect; add isinstance(...) checks if you want enforcement.
- enumerate(list) gives (index, value) pairs.
- Use .strip() before .isdigit() to handle spaces (e.g., ' 5 ').
- Return None from helpers to signal failure; print messages in the caller.

Next lesson: Week 1 - Lesson 6 wrap-up or move to Week 2 tooling (Git, venv, pytest, logging) depending on your pace.