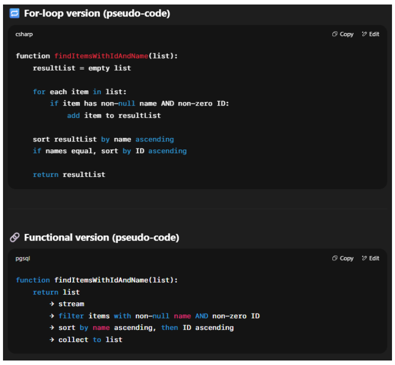
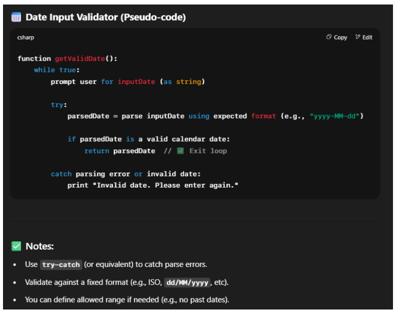
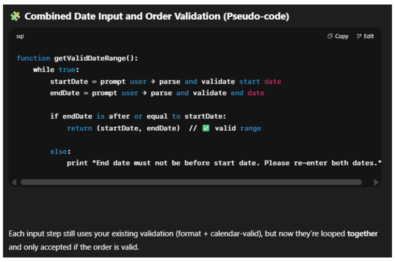
SE1919 - SE200055 - Nguyễn Quang Khiêm

ChatGPT’s Use and Implementation.

Pseudo-code generation & algorithm analysis..

	For sorting and returning a list	For generating a date-entry validator	For combining date-entry and verification modules
Prompt	I am looking to implement a function which returns all items which have an ID and name within a list. I want the list to be sorted alphabetically and then sorted by ascending ID if the names match. Give me both the for-loop, and the functional chain that does the same thing. Do not mutate the original list. Give me the pseudo-code template for both of those.	I want to make a function to take an input until it satisfies certain conditions only then would it be allowed to break. I am making one for a date passed in as a string, and the job of the verifier is to scan the input. And parse the input for an appropriate date which follows a certain format. If the format isn't met or the date is faulty ie not a day in a year then ask the user for reentry. Pseudo-code again.	I have three separate functions, one which takes in the starting date of something, another for the end date, and then the third one which takes in both of those dates and verifies whether they are in the correct order, i.e. the end date doesn't come before the start date. Couple them together in a single function to take in both dates, and ask the user to reenter both if the order is not logically correct.
Pseudo-code			

Pseudo-code

For-loop version (pseudo-code)

```
function findItemsWithIdAndName(list):
    resultList = empty list

    for each item in list:
        if item has non-null name AND non-zero ID:
            add item to resultList

    sort resultList by name ascending
    if names equal, sort by ID ascending

    return resultList
```

Functional version (pseudo-code)

```
function findItemsWithIdAndName(list):
    return list
        → stream
        → filter items with non-null name AND non-zero ID
        → sort by name ascending, then ID ascending
        → collect to list
```

Date Input Validator (Pseudo-code)

```
function getValidDate():
    while true:
        prompt user for inputDate (as string)

        try:
            parsedDate = parse inputDate using expected format (e.g., "yyyy-MM-dd")

            if parsedDate is a valid calendar date:
                return parsedDate  // ✅ Exit loop

        catch parsing error or invalid date:
            print "Invalid date. Please enter again."
```

✅ **Notes:**

- Use `try-catch` (or equivalent) to catch parse errors.
- Validate against a fixed format (e.g., ISO, `dd/MM/yyyy`, etc).
- You can define allowed range if needed (e.g., no past dates).

Combined Date Input and Order Validation (Pseudo-code)

```
function getValidDateRange():
    while true:
        startDate = prompt user → parse and validate start date
        endDate = prompt user → parse and validate end date

        if endDate is after or equal to startDate:
            return (startDate, endDate)  // ✅ valid range

        else:
            print "End date must not be before start date. Please re-enter both dates."
```

Each input step still uses your existing validation (format + calendar-valid), but now they’re looped **together** and only accepted if the order is valid.

	For sorting and returning a list	For generating a date-entry validator	For combining date-entry and verification modules
Code analysis	🔍 **Algorithm Analysis: Filter and Sort Items by Name and ID** **Purpose:** Return a new list of items that have a non-zero ID and a non-empty name. The list is sorted alphabetically by name, then by ID (ascending) if names match. **Logic:** 1. Filter items where `id ≠ 0` and `name` is not null/empty. 2. Sort by `name` ascending, then `id` ascending. 3. Return a new list (original list remains unchanged).	🔍 **Algorithm Analysis: Valid Date Input Loop** **Purpose:** Continuously prompt the user until they enter a valid date string in the correct format (e.g., `"yyyy-MM-dd"`), and the date is a real, valid calendar date. **Logic:** 1. Loop until input is valid 2. Try parsing the input with a strict formatter 3. If parsing succeeds → return the date 4. If it fails → prompt again **Edge Cases:**	🔍 **Algorithm Analysis: Valid Date Range Input** **Purpose:** Prompt the user to enter a valid **start** and **end date**. Ensure: ● Each date is valid and parsable ● **The end date is not before the start date** **Logic:** 1. Prompt and validate `startDate` 2. Prompt and validate `endDate` 3. If `endDate ≥ startDate` → return both **Edge Cases:**

	For sorting and returning a list	For generating a date-entry validator	For combining date-entry and verification modules
	Edge Cases: • Empty input → returns empty list • Null/empty name → skipped • Same name → sorted by ID	• Invalid formats (`"12/31/2023"` or `"2023-13-40"`) → rejected • Blank or null input → rejected • Leap year or invalid day/month → caught during parsing	• Invalid format or impossible date → rejected • End date before start date → rejected • Equal dates → accepted (if allowed by logic)