Q1: The Smart Home Sensor

The Setup: You are programming a logic controller for a smart home. The system receives data from various sensors and needs to store them in the correct format to trigger actions.

Your Tasks:

1. **Numeric Types:** Store the current room temperature as a `float` (e.g., `22.5`) and the number of people in the room as an `int`.
2. **Boolean:** Create a variable `is_motion_detected`. If `people_count` is greater than 0, set this to `True`; otherwise, `False`.
3. **String Manipulation:** The system generates a status message. Create a string `status = "System Active"`. Use a string method to convert it to all uppercase.
4. **Casting:** You receive a sensor reading as a string `"19"`. Convert it to an `int` so you can add it to the `current_temperature`.

Q2: The E-Commerce Cart

The Setup: A customer is shopping on your website. You need to manage their "collection" of items. This scenario practices **Lists**, **Tuples**, and **Dictionaries**.

Your Tasks:

1. **List:** Create a list called `cart_items` containing three strings: `"Laptop"`, `"Mouse"`, and `"USB Cable"`. Add `"Keyboard"` to the end of the list.
2. **Tuple:** Store the dimensions of the laptop box (length, width, height) in a `tuple` called `box_dimensions`. *Challenge: Try to change one value in the tuple and observe the error.*
3. **Dictionary:** Create a dictionary `product_info` for the "Laptop". It should have keys for `"brand"`, `"price"`, and `"in_stock"` (a boolean).
4. **Set:** You have a list of categories: `["Electronics", "Office", "Electronics", "Gadgets"]`. Convert this list into a `set` to remove the duplicates.

Q3: The Library Management System

The Setup: You are organizing a small digital library. This helps you practice more complex data structures and type-specific operations.

Your Tasks:

1. **Nested Data:** Create a list of dictionaries called `library`. Each dictionary should represent a book with keys: `"title"`, `"author"`, and `"year"`.

2. **Logic & Types:** The library adds a "unique ID" to every book. Use the `range()` function to generate IDs from 100 to 105 and store them in a list.
3. **String Formatting:** Print the first book's details using an f-string: *"The book {title} was written by {author}."*
4. **Type Checking:** Use the `type()` function to verify that your `library` variable is indeed a `list`.

Q4: The RPG Character Creator

The Setup: You are designing the backend for a fantasy Role Playing Game (RPG). You need to store complex player stats that change as the game progresses.

Your Tasks:

1. **Dictionary:** Create a dictionary named `player_stats`. Include:
   - `"name"` (string)
   - `"level"` (int)
   - `"inventory"` (a list of strings: `"sword"`, `"shield"`, `"potion"`)
2. **Updating Nesting:** The player finds a `"dragon scale"`. Append this new item to the list *inside* the dictionary.
3. **The "Level Up" Math:** The player leveled up! Increase the `level` value in the dictionary by 1.
4. **Tuple Protection:** Store the player's starting coordinates (X, Y) in a tuple called `spawn_point`. Explain why a tuple is better than a list for a spawn point.

Q5: The Weather Station Logger

The Setup: You have a device that records temperatures every hour. You need to handle data cleanup and summary.

Your Tasks:

1. **List to Set:** You have a list of hourly readings: `[22, 24, 22, 21, 24, 25, 22]`. Convert this to a `set` to find all the **unique** temperatures recorded that day.
2. **Boolean Logic:** Create a variable `is_freezing`. Use a comparison operator to set it to `True` if the minimum value in your list is below 0.
3. **Complex String:** Create a variable `report`. Use `.join()` to turn your list of unique temperatures (converted to strings) into a single string separated by commas.

Q6: The Restaurant Menu (Nested Logic)

The Setup: A digital menu needs to categorize items by "Appetizers," "Mains," and "Desserts."

Your Tasks:

1. **Nested Dictionary:** Create a dictionary called `menu`.
   - The keys should be categories (e.g., `"Mains"`).
   - The values should be another dictionary where the key is the dish name and the value is the price.
2. **Accessing Data:** Write a print statement that pulls the price of a specific dish from deep inside the `menu` dictionary.
3. **Type Discovery:** Use `isinstance()` to check if the value of the `"Mains"` key is a dictionary.

Let's push into the "Professional Grade" scenarios. These assignments focus on **Data Integrity**, **State Management**, and handling **Real-World API-style data structures**.

Q7: The Banking Transaction Log

The Setup: You are processing a stream of financial data. Precision and history are key here. You need to distinguish between the *action* and the *amount*.

Your Tasks:

1. **List of Tuples:** Create a variable `transactions`. Store at least four transactions as tuples, like: `("deposit", 500.25), ("withdrawal", 100.00).`
2. **Immutability Logic:** Why did we use a **tuple** for each transaction instead of a **list**? (Hint: Can a transaction amount be changed after it happens?)
3. **Float Precision:** Create a `current_balance`. Calculate the sum of your transactions.
4. **String Formatting:** Use a "f-string" to display the balance formatted to 2 decimal places (e.g., `$400.25`).

Q8: The Social Media "Like" Tracker

The Setup: You're building the backend for a post's engagement. You need to ensure that one user cannot "Like" a post multiple times.

Your Tasks:

1. **Set:** Create a set called `user_ids_who_liked`. Add three unique strings (usernames).
2. **Duplicate Handling:** Try to `.add()` a username that is already in the set. Check the length of the set—did it change?
3. **Type Conversion:** You need to display these users alphabetically. Convert the `set` into a `list` and use the `.sort()` method.

4. **Boolean Check:** Create a variable `has_liked`. Use the `in` keyword to check if `"alpha_user"` is in your set.

Q9: The Global Shipping Manifest

The Setup: You are managing a shipping container. You have a mix of data: the destination (constant), the item list (changeable), and the weight (numeric).

Your Tasks:

1. **The Master Object:** Create a dictionary called `container`.
   - `id`: An integer.
   - `destination`: A string.
   - `contents`: A list of dictionaries (each dictionary should have `"item_name"` and `"weight"`).
2. **Deep Access:** Access the weight of the *second* item in the `contents` list.
3. **NoneType:** Create a variable `customs_clearance_date` and set it to `None`. This represents that the event hasn't happened yet.
4. **Type Validation:** Write a script that checks if `container["id"]` is an integer using `type()`.