Problem Title: Student Grade Categorizer (if...else if...else)

Input * Variable: score

- Type: Numeric (0 to 100)

Scenario A teacher needs to assign a letter grade based on a numeric score.

- If the score is 90 or above, the grade is "A".
- If the score is between 80 and 89, the grade is "B".
- If the score is between 70 and 79, the grade is "C".
- Anything below 70 is a "Fail".

Output * A printed string: "The student's grade is: [Grade]"

Problem Title: Daily Sales Logger (for loop)

Input * Variable: daily_sales

- Type: Numeric Vector (e.g., c(1200, 450, 900, 1500))

Scenario A shop owner wants to see the total sales processed at the end of a 4-day period. Use a loop to visit each value in the vector and add it to a running total starting at zero.

Output * A printed message: "Final total sales: [Total Value]"

Problem Title: Inventory Stock Alert (for loop + next)

Input * Variables: item_names (Character Vector), stock_levels (Numeric Vector)

- Constraints: The vectors are of equal length.

Scenario A warehouse manager is auditing stock. You need to loop through the inventory.

- If an item has more than 10 units, it is considered "Sufficient" and should be skipped.
- If an item has 10 units or fewer, print a reorder alert.

Output * Printed alerts only for low-stock items: "Alert: [Item Name] is low on stock ([Count] left)."

Problem Title: Smart Home Temperature Regulator (while loop)

Input * Variable: current_temp (Initial value: 30)

- Variable: target_temp (Constant: 22)

Scenario An AC unit is cooling a room. As long as the current_temp is higher than the target_temp, the AC stays on. In each cycle of the loop, the temperature drops by 1.5 degrees. The loop should stop once the target is reached or exceeded.

Output * The temperature at each step and a final message: "Target reached. Current temp: [Value]"

Problem Title: ATM Cash Dispenser Validation (while + break + if...else)

Input * Variable: account_balance (e.g., 5000)

- Variable: withdrawal_request (e.g., 200)
- Variable: atm_cash_available (e.g., 1000)

Scenario A customer wants to keep withdrawing the same withdrawal_request amount repeatedly in a single session.

- The loop should continue until the customer chooses to stop (simulate this with a set number of iterations).
- **Condition 1**: If the account_balance falls below the request, print "Insufficient Funds" and break.

- **Condition 2**: If the atm_cash_available falls below the request, print "ATM out of cash" and break.
- Otherwise, subtract the amount from both the balance and the ATM cache and print the remaining balance.

Output * Detailed log of each withdrawal and a final reason for stopping the session

---

Problem Title: Missing Data Data-Cleaner (Nested loops + if)

Input * Object: A Data Frame with 5 rows and 3 columns (e.g., Name, Age, Salary).

- Constraints: Some cells contain NA (missing values).

Scenario You are preparing a report for HR. You must check every single cell in the table (row by row, and column by column).

- If a cell contains a value, do nothing.
- If a cell contains NA, print the exact location (Row Number and Column Name) so the HR team can find and fix it.

Output * List of missing data points: "Missing data found at Row [X], Column [Y]"

Problem Title: E-Commerce Shipping Tier Calculator (if...else if...else)

Input * Variable: order_value (Numeric)

- Variable: customer_location (Character: "Local", "National", "International")

Scenario A shipping software must determine the delivery fee based on two factors:

- If the customer_location is "Local", shipping is always **0**.
- If "National":
    - Orders over **100** get free shipping.
    - Orders **100 or below** cost **10**.
- If "International":
    - Orders over **500** cost **20**.
    - Orders **500 or below** cost **50**.

Output * A printed message: "Shipping cost for your [Location] order is: [Cost]"

Problem Title: Monthly Subscription Billing (for loop + if...else)

Input * Variable: usage_minutes (Numeric Vector representing 10 different users)

- Constraints: The base plan allows for **500 minutes**.

Scenario A telecom company bills users a flat rate of **$30**. However, if a user exceeds the 500-minute limit, they are charged an additional **$0.10** for every minute over the limit. Loop through the vector of usage minutes to calculate the final bill for each user.

Output * For each user, print: "User [Index]: Total Bill is $[Amount]"

Problem Title: Financial Goal "Runway" Simulator (while loop)

Input * Variable: savings_account (Initial: 10,000)

- Variable: monthly_expenses (Initial: 1,500)
- Variable: inflation_rate (1.02 per month)

Scenario A financial planner wants to show a client how long their savings will last if they stop working.

- The loop should run as long as savings_account is greater than monthly_expenses.
- Each month, the monthly_expenses increase by the inflation_rate (multiply by 1.02).
- Subtract the new expense from the savings.

- Count how many months pass before the account is depleted.

Output * A final statement: "Your savings will last for [Month Count] months."

Problem Title: Quality Control Batch Tester (Nested loops + break)

Input * Object: A List of 5 "Batches". Each batch is a Numeric Vector of 10 "Sensor Readings".

- Constraints: Readings should be between **20.0 and 25.0**.

Scenario You are auditing a factory line. You must check every sensor reading in every batch.

- Use an outer loop for the batches and an inner loop for the readings.
- If a single reading in a batch is outside the allowed range (e.g., 18.5 or 26.1):
    - Print a "Batch [X] Rejected" message.
    - Use a **break** to stop checking the rest of the readings in *that specific batch* and move immediately to the next batch.

Output * A log showing which batches passed and which were rejected at the first sign of error.

Problem Title: Selective Data Aggregator (for loop + next + if)

Input * Variable: transaction_amounts (Numeric Vector including positive and negative values)

- Variable: transaction_types (Character Vector: "Sales", "Refund", "Adjustment")

Scenario A bookkeeper needs a total of all **successful sales**.

- Loop through the transactions.
- If the transaction_type is "Refund" or "Adjustment", use **next** to skip the calculation.
- If the transaction_amount is negative or zero (even if labeled "Sales"), skip it.
- Otherwise, add the amount to a total_revenue variable.

Output * "The total verified revenue is: [Total]"

Problem Title: Dynamic Loyalty Point Multiplier (if...else if...else inside for)

Input * Variable: purchase_history (Numeric Vector of transaction amounts)

- Variable: membership_tier (Character: "Gold", "Silver", "Bronze")

Scenario A retail store rewards points based on both the amount spent and the membership level. Use a loop to process each purchase:

- **Gold members**: Receive 3 points for every $1 spent.
- **Silver members**: Receive 2 points for every $1 spent.
- **Bronze members**: Receive 1 point for every $1 spent.
- **Logic**: If a purchase is over $500, the user gets an extra 50 bonus points regardless of tier.

Output * A summary for each transaction: "Purchase $[Amount]: Earned [Points] points."

- The grand total of points earned for the entire history.

Problem Title: Temperature Sensor "Warm-up" Monitor (while loop)

Input * Variable: sensor_reading (Initial: 15.0)

- Variable: threshold (Constant: 25.0)

Scenario A laboratory incubator must reach a stable temperature before an experiment starts.

- The system checks the temperature every "minute" (loop iteration).
- In each iteration, a random decimal between 0.1 and 2.0 is added to the sensor_reading.
- The loop must continue as long as the reading is below the threshold.

Output * A log for every minute: "Minute [X]: Temperature is [Value]°C."

- A final message: "Incubator Ready."

Problem Title: High-Frequency Trading "Safety Switch" (for + break)

Input * Variable: stock_prices (Numeric Vector of prices recorded every second)

- Variable: max_loss_limit (Numeric: e.g., -500)

Scenario An automated trading bot is tracking profit/loss.

- The bot starts with a current_pnl (Profit and Loss) of 0.
- It loops through the stock_prices vector. For every price change (current price minus previous price), it updates the current_pnl.
- **The Safety Switch**: If the current_pnl ever drops below the max_loss_limit, the bot must "kill" the operation immediately to prevent further bankruptcy.

Output * A report of the P&L at each second.

- If triggered: "Emergency Stop! Loss limit exceeded at Second [X]."
- If not triggered: "Trading session completed successfully.

Problem Title: Automated Payroll Audit (Nested loops + next + if)

Input * Object: A List of 4 "Departments." Each department contains a Numeric Vector of "Employee Hours" for the week.

- Constraints: A standard workweek is 40 hours.

Scenario An auditor is looking for "Extreme Overtime" (over 60 hours).

- Loop through each department (Outer Loop).
- Loop through each employee's hours (Inner Loop).
- **Condition 1**: If an employee worked 40 hours or less, use next to ignore them (they are standard).
- **Condition 2**: If an employee worked between 41 and 60 hours, print "Moderate Overtime detected."
- **Condition 3**: If an employee worked over 60 hours, print a high-priority "Audit Required" message.

Output * A categorized list of overtime alerts for each department.

Problem Title: Password Strength Validator (while + if...else + break)

Input * Variable: user attempts (A Character Vector of passwords a user is trying to set).

- Constraints: A valid password must be at least 8 characters long.

Scenario A security system is forcing a user to set a password during account creation.

- The system loops through the user attempts.
- If a password is less than 8 characters, it prints "Too short" and moves to the next attempt.

- If a password is 8 characters or more, it prints "Password accepted" and **breaks** the loop (the user is finished).
- If the user runs out of attempts (the end of the vector) without a valid password, print "Account Locked."

Output * Status updates for each attempt until the user succeeds or is locked out.

Problem Title: The Safety Countdown (repeat + break) Input * Variable: countdown_start (Numeric: e.g., 5)

Scenario A rocket launch sequence is initiated.

- Use a repeat loop to print the current countdown number.
- In each iteration, decrease the number by 1.
- **Rule**: The loop must stop when the number reaches 0.
- Use a break statement to exit and print "Blast off!".

Output * Sequence: "5... 4... 3... 2... 1... Blast off!"

Problem Title: The "Double or Nothing" Gambling Simulation Input * Variable: wallet (Initial: 100)

- Variable: goal (Constant: 500)

Scenario A player is at a casino playing a game where they either double their money or lose it all.

- Use a repeat loop to simulate rounds of betting.
- In each round, use a random function (like flipping a coin) to either:
    1. Double the current wallet value.
    2. Set the wallet to 0.
- **Exit Condition 1**: If the wallet reaches or exceeds the goal, print "Goal Reached!" and break.
- **Exit Condition 2**: If the wallet hits 0, print "Bankrupt!" and break.

Output * A log of the wallet balance after each round.

- A final status message explaining why the loop stopped

Problem Title: Newton's Method for Square Roots (Numerical Approximation) Input * Variable: number (The value to find the square root of, e.g., 25)

- Variable: guess (Initial guess, e.g., 1)
- Variable: tolerance (Constant: 0.00001)

Scenario In mathematics, you can find a square root by repeatedly refining a guess.

- Use a repeat loop to calculate a new_guess using the formula: $$new\_guess = \frac{(guess + \frac{number}{guess})}{2}$$
- **Logic**: Calculate the difference between the old_guess and the new_guess.
- **Exit Condition**: If the absolute difference is less than the tolerance, the answer is accurate enough. Use break.
- **Update**: If not accurate enough, make the guess equal to the new_guess and repeat.

Output * The approximated square root.

- The number of iterations it took to get there