**Basic Calculator with Conditional Checks**

**Task**: Write a shell script that acts as a simple calculator.

**Requirements**:

Prompt the user for two numbers and an operator (e.g., +, -, \*, /).

Perform the calculation using appropriate arithmetic operators.

Use conditionals to handle invalid operator input and division-by-zero errors.

Include a loop that allows the user to perform multiple calculations in one session.

Provide clear error messages and usage prompts.

**User Registration Script (Variables and Loops)**

**Task**: Develop a script that collects user information and stores it in a file.

**Requirements**:

Prompt for a username, email, and password.

Validate that the email follows a simple pattern (e.g., something@domain).

Use loops to allow multiple user entries in one execution.

Print a summary of all users entered.

Store user data in a text file, appending new entries each time the script runs.

**File Organizer Using Conditionals and Loops**

**Task**: Create a script that sorts files into different folders based on file extensions.

**Requirements**:

Prompt for a source directory.

Check if it exists; if not, display an error.

Use loops to iterate over all files, checking file extensions (e.g., .txt, .jpg, .sh).

Use conditionals to move files into corresponding folders (e.g., Text\_Files, Images, Scripts).

Include an option to create new categories if needed.

**Interactive Quiz with Conditionals and Arrays**

**Task**: Write a script that presents a small quiz to the user.

**Requirements**:

Store questions and multiple-choice answers in arrays.

Prompt the user for responses, use conditionals to check correctness, and keep a score.

After all questions are answered, display the user’s total score and percentage.

Implement a loop so the user can retake the quiz if desired.

**Simple Inventory Management (Loops, Conditionals, File I/O)**

**Task**: Build a script that simulates basic inventory operations.

**Requirements**:

Use a text file to store item name, quantity, and price.

Provide a menu with options: **Add New Item**, **Update Item**, **View All Items**, **Delete Item**.

Use loops to keep showing the menu until the user chooses to exit.

Implement conditionals to handle edge cases (e.g., updating non-existent items).

**System Resource Checker (Basic Debugging)**

**Task**: Write a script to check and log CPU, memory, and disk usage.

**Requirements**:

Use commands like free, df, and grep to retrieve usage data.

Store the output in a log file with timestamps.

Implement a debug mode (e.g., ./syscheck.sh --debug) that prints additional diagnostic messages (use set -x or manual debug prints).

Include conditionals to alert if usage exceeds certain thresholds (e.g., disk usage > 80%).

**File Comparison Tool with Regex**

**Task**: Create a script that compares two text files and highlights matching or differing lines.

**Requirements**:

Prompt for two file paths, validate their existence.

Use regular expressions to find lines that match a certain pattern (e.g., lines containing an email or IP address).

Print a summary of how many matching lines were found in each file.

Provide an option to output these matching lines to a new file.

**Text-Based Adventure Game (Loops, Conditionals, Functions)**

**Task**: Write a mini interactive game in shell script.

**Requirements**:

Use functions to handle different “rooms” or “scenes” of the game.

Store game state (inventory items, player health) in variables.

Use loops and conditionals to navigate the user through multiple choices (e.g., “go left,” “go right,” “pick up item”).

End the game with a success or failure message based on user choices.

**Folder Backup Script with Error Handling**

**Task**: Develop a script that backs up a chosen folder to a compressed archive.

**Requirements**:

Prompt the user for the source folder and a backup location.

Use tar and gzip (or zip) to create a compressed archive.

Check for errors (e.g., invalid path, insufficient permissions) and log them to a file.

Include a restore option to extract the backup to a chosen location.

**Menu-Driven Shell Script with Parameter Passing**

**Task**: Create a menu-driven script that supports command-line parameters to control certain functions.

**Requirements**:

Provide a menu with at least three functionalities (e.g., system info, user management, file search).

If a parameter is passed (e.g., --auto), run one of the functionalities automatically.

Use functions for each menu option, and handle incorrect parameter usage gracefully.

Log all actions to a file for auditing.