Scratch programming Project Assignment 1 in classroom.

(1) **PROBLEM STATEMENT OF THE HABBIT TRACKER**

Problem Statement: Habit Tracker

In today’s fast-paced world, maintaining consistency in personal development and healthy habits can be challenging. Many individuals struggle to keep track of their daily habits, which hinders their progress toward achieving their goals. Current solutions often lack personalization, user engagement, and effective reminders, leading to decreased motivation and accountability.

Objective:
To develop a user-friendly habit tracker application that helps users set, track, and maintain their habits effectively. The app should provide personalized recommendations, progress visualizations, and reminders to enhance user engagement and accountability.

Key Features:

1. **User Profiles:** Allow users to create and customize profiles to set specific habit goals.
2. **Habit Tracking:** Enable users to log daily progress and view historical data.
3. **Reminders and Notifications:** Provide customizable reminders to prompt users to maintain their habits.
4. **Gamification:** Introduce rewards and challenges to motivate users and make tracking enjoyable.
5. **Insights and Analytics:** Offer visual reports and insights on progress to help users stay focused and adjust strategies as needed.

Target Users:
Individuals seeking to improve their daily routines, whether for health, productivity, or personal growth.

Q NO (2) ALGORITHM OF HABBIT TRAKER

Algorithm for Habit Tracker Project in Scratch

1. **Setup Variables**:
    - Create variables for each habit you want to track (e.g., "Exercise", "Read", "Meditate").
    - Create a variable to store the count of completed days for each habit (e.g., "Exercise Days", "Read Days").
2. **User Interface**:
    - Design a user-friendly interface with buttons for each habit (e.g., "I Did It!" button).
    - Add a display area to show the number of days completed for each habit.
    - Include a reset button to clear counts if desired.
3. **Initialize Variables**:
    - Set all habit count variables to zero when the project starts.
4. **Track Habit Completion**:
    - When the user clicks the button for a specific habit:
        - Increase the corresponding habit count variable by 1.
        - Update the display to show the new count.
5. **Optional Features**:
    - **Weekly View**: Allow users to input whether they completed the habit each day of the week.
    - **Streak Tracking**: Keep track of how many consecutive days a habit was completed.
    - **Reminders**: Create a simple reminder system to prompt users to complete their habits.
6. **Save Data**:
    - Use the “cloud” feature (if applicable) to save the counts so they persist between sessions.
7. **Feedback**:

- Provide positive feedback or rewards (e.g., animations, sounds) when users click to mark a habit as completed.

8. **End of Day Summary**:
    - Optionally, create a summary at the end of the day that shows total completed habits and encourages the user.

Implementation Steps

1. **Create Sprites**: Make sprites for each habit, buttons, and the display area.
2. **Scripts for Buttons**: Write scripts for each button to increment the respective count variable.
3. **Update Display**: After each button press, update the display area to reflect the current count.
4. **Testing**: Test each feature to ensure everything works as expected.
5.

(2) ASSIGNMENT OF THE SIMPLE TO DO LIST

(1) PROBLEM STATEMENT OF THE SIMPLE TO DO LIST:-

Key Features:

1. **Add Tasks:**
    - Users should be able to input a new task through a text box and a button.
    - The new task should be displayed in a list.
2. **View Tasks:**
    - The current list of tasks should be visible on the screen.
    - Each task should be displayed clearly, with an option to mark it as complete.
3. **Remove Tasks:**

- Users should have the ability to remove a task from the list once it is completed or no longer needed.
- This can be done via a "Delete" button next to each task.

4. **Clear List:**
    - Include an option to clear all tasks from the list.

User Interface Requirements:

- A simple and intuitive layout with buttons for adding, removing, and clearing tasks.
- Clear instructions or labels for each action.

Target Users:

- Individuals looking for a simple way to organize their daily tasks.

Technical Constraints:

- The project should be implemented using Scratch's visual programming blocks.
- Ensure that the program runs smoothly and responds to user inputs.

ALGORITHM OF THE SIMPLE TO DO LIST :-

1. Setup

- **Create a new Scratch project.**
- **Add a list called `To Do List`.**

2. Variables

- Create a variable called `Input` to store user input.
- Create a variable called `Index` to track the position in the list.

3. Start the Program

1. **When the green flag is clicked:**
    - Clear the `To Do List`.

- Set `Index` to 1 (this represents the first item).

4. Input Handling

1. **Ask for input:**
   - Use the `ask "What do you want to add to your to-do list?"` block.
2. **Wait for user response:**
   - Store the answer in the `Input` variable.

5. Add to the List

1. **Add the input to the list:**
   - Use the `add (Input) to To Do List` block.
2. **Reset Input:**
   - Set `Input` to an empty string (optional for clarity).

6. Display the List

1. **Say the current list:**
   - Use a loop to display all items in `To Do List`:
     - Repeat until `Index` is greater than the length of `To Do List`:
       - Say (item (Index) of `To Do List`).
       - Change `Index` by 1.

7. Loop for Continuous Input

1. **After displaying the list, go back to step 4:**
   - Ask for input again.