7a Theory JS API pages 105 - 113 Creating A Hangman Game Part I

Nicholas Bruzzese

July 27, 2025

1 Understanding Loops in JavaScript for a Hangman Game

Loops are a fundamental concept in programming, allowing a program to repeat a set of instructions multiple times. In the context of creating a Hangman game using JavaScript, loops are essential for managing the repetitive tasks that occur during gameplay. This lecture explains the theory of loops as they relate to the Hangman game, focusing on their purpose, structure, and application in a way that is accessible to a young learner with some JavaScript experience.

1.1 The Role of Loops in the Hangman Game

In the Hangman game, the program must perform several tasks repeatedly until the player either guesses the word or chooses to quit. These tasks include showing the current state of the word (with blanks for unguessed letters), getting a guess from the player, checking if the guess is valid, updating the game state, and determining if the game is complete. Since these steps need to happen multiple times, and the exact number of repetitions depends on the player's guesses, a loop is the ideal tool to manage this process.

A loop allows the program to cycle through these tasks until a specific condition is met, such as the player guessing all the letters in the word. In the Hangman game, the loop ensures that the game continues to prompt the player for guesses and update the game state until the word is fully revealed or the player decides to stop.

1.2 The Game Loop Concept

The Hangman game uses a structure called a **game loop**, which is common in many computer games. A game loop is a repetitive process that handles three main tasks:

1. **Taking input from the player**: In Hangman, this involves using a JavaScript `prompt` to ask the player for a letter guess.
2. **Updating the game state**: This means updating the array that tracks the player's progress (e.g., filling in correct letters or noting incorrect guesses).
3. **Displaying the current state**: The program shows the player the current state of the word, with blanks for unguessed letters and filled-in letters for correct guesses.

This loop runs continuously, repeating these steps in each iteration, until the game ends. In the Hangman game, the loop stops when the player has guessed all the letters in the word, at which point the program displays a congratulatory message.

1.3 The While Loop in Hangman

The specific type of loop used in the Hangman game is a **while loop**. A while loop in JavaScript repeats a block of code as long as a certain condition is true. In the game's pseudocode, the while loop is described as:

```
While the word has not been guessed {
  Show the player their current progress
  Get a guess from the player
  If the player wants to quit the game {
    Quit the game
  }
  Else If the guess is not a single letter {
    Tell the player to pick a single letter
  }
  Else {
    If the guess is in the word {
      Update the player's progress with the guess
    }
  }
}

```

This pseudocode outlines the logic of the while loop. The condition "while the word has not been guessed" means the loop continues as long as there are still letters in the word that the player has not correctly guessed. The loop performs the following steps in each iteration:

- Displays the current state of the word (e.g., `_ _ _ _` for a four-letter word like "fish").
- Prompts the player for a guess using a `prompt` dialog.
- Checks if the player wants to quit (e.g., by clicking Cancel, which returns `null`).
- Validates the guess to ensure it is a single letter.
- Updates the game state if the guess is correct by filling in the appropriate blanks.
- Continues looping until all letters are guessed.

The while loop is ideal for the Hangman game because the number of guesses is unknown in advance—it depends on how quickly the player guesses the word. Unlike a loop that runs a fixed number of times, a while loop can adapt to the player's progress.

1.4 Tracking Progress in the Loop

To determine when the word has been guessed, the game uses a variable to track the number of remaining letters that need to be guessed. This variable starts with the total number of letters in the word and decreases by one for each correct letter guessed. For

example, if the word is "fish" (four letters), the variable starts at 4. If the player guesses "i", and it appears once, the variable decreases to 3. When this variable reaches 0, the loop ends because the word has been fully guessed.

The game also uses an **answer array** to represent the word's current state. This array starts with a blank space (represented as `_`) for each letter in the word. For "fish", the array would initially be `["_ ", "_ ", "_ ", "_ "]`. When the player guesses "i", the array updates to `["_ ", "i ", "_ ", "_ "]`. This array is displayed to the player in each loop iteration to show their progress.

1.5 Why Loops Are Important

Loops are crucial in the Hangman game because they allow the program to handle the repetitive nature of gameplay efficiently. Without a loop, the programmer would need to write the same code multiple times, which would be impractical and error-prone. The while loop simplifies the code by encapsulating the repetitive tasks in a single block that runs until the game is complete.

Additionally, loops make the game dynamic. Since the number of guesses varies depending on the player's skill, a loop allows the program to adapt to different scenarios without requiring a fixed number of steps. This flexibility is a key feature of programming, enabling games like Hangman to respond to user input in real time.

1.6 Loops and User Interaction

The while loop in the Hangman game integrates with JavaScript's user interaction functions, such as `prompt`, `confirm`, and `alert`, to create an interactive experience. For example:

- The `prompt` function is used within the loop to get the player's guess in each iteration.
- The `alert` function may be used to display the current state of the word or to inform the player if their guess is invalid (e.g., not a single letter).
- The `confirm` function could be used to ask if the player wants to quit, allowing the loop to exit early if they choose to stop.

These functions pause the JavaScript interpreter until the player responds, ensuring that the loop waits for user input before proceeding to the next iteration. This pause is important because it gives the player time to read messages and make decisions, making the game feel interactive and responsive.

1.7 Pseudocode and Loop Design

Before writing the actual JavaScript code for the Hangman game, the program is planned using **pseudocode**, a simplified way of describing the program's logic in plain English mixed with code-like structures. The pseudocode for the Hangman game includes the while loop, as shown earlier, and helps outline how the loop will work before dealing with the technical details of JavaScript syntax.

Pseudocode is not executable by a computer but serves as a blueprint for the program. It helps programmers think through the loop's structure, ensuring that all necessary steps

(like validating guesses and updating progress) are included. By designing the loop in pseudocode first, the programmer can focus on the game's logic without getting bogged down in coding details.

1.8 Practical Example of the Loop

To illustrate how the while loop works in the Hangman game, consider the word "fish". The loop operates as follows:

1. **Initial State**: The answer array is `["_ ", "_ ", "_ ", "_ "]`, and the remaining letters variable is 4.
2. **First Iteration**: The player sees `_ _ _ _` and guesses "i". The program checks that "i" is a valid single letter and is in "fish". The answer array updates to `["_ ", "i ", "_ ", "_ "]`, and the remaining letters variable decreases to 3. The loop continues because the word is not fully guessed.
3. **Second Iteration**: The player sees `_ i _ _` and guesses "s". The program updates the array to `["_ ", "i ", "s ", "_ "]`, and decreases the remaining letters to 2. The loop continues.
4. **Subsequent Iterations**: The loop repeats, updating the array and variable until the player guesses "f" and "h", making the array `["f ", "i ", "s ", "h "]`, and the remaining letters 0. The loop ends, and the program congratulates the player.

This example shows how the while loop manages the game's flow.

System: