Evaluation of the Setup

1. **Board Representation**
   - Represent the 3×3 board as a *flat list of length 9*.
   - Each square is initially labeled with numbers 1–9, which serve as identifiers for available moves.
   - This avoids needing a separate "valid moves" list—open squares are simply those that still contain their original number.
2. **Move Encoding**
   - Player X → `10`
   - Player O → `-10`
   - This choice makes the board easy to check mathematically. Instead of holding text, it holds numbers that allow quick calculation.
3. **Printing**
   - When displaying the board, you translate:
     - `10 → "X"`
     - `-10 → "O"`
     - Any other value → `" "`
   - This keeps the internal representation numeric for logic, but user-friendly when shown.

Computational Thinking Elements

1. **Abstraction**
   - We separated the *internal state* (numbers in a list) from the *presentation layer* (printing X's and O's).
   - Players never see the raw data; they see symbols. That's a clean abstraction barrier.
2. **Representation with Numbers**
   - Instead of storing strings like `"X"` or `"O"`, you use integers.
   - This allows you to use arithmetic and sums for checking win conditions:
     - If the sum of a row/column/diagonal = `30` → X wins (3 × 10).
     - If the sum = `-30` → O wins (3 × -10).
   - This is an elegant computational trick: *using math to simplify logic*.
3. **Decomposition**
   - The game is broken into components:
     - **State management**: list of numbers.
     - **Update rule**: replace number with `10` or `-10`.
     - **Printing logic**: map numbers to human symbols.
     - **Win check**: numeric sum of board segments.
4. **Algorithmic Thinking**
   - We've designed a process that's repeatable:
     - Pick move → update board → check winner → print board.

    - The encoding of moves as integers allows efficient winner-check algorithms.

5. **Pattern Recognition**
    - We recognized that using `10` and `-10` creates a clear mathematical pattern (multiples of 10) that’s easy to evaluate with addition.

This design is **computationally elegant** because it reduces complexity by representing game state in numbers instead of strings, which enables arithmetic shortcuts for win detection while keeping the user-facing side simple.