**Rust Programming Practice Tasks**

1. **Variable Shadowing with Types**  
   Task: Write a program where you define a variable distance initially as an integer (representing meters). Then, shadow this variable by converting the distance to kilometers (floating-point). Print both values using the same variable name (distance) to demonstrate type shadowing.
2. **Ownership and Returning Ownership**  
   Task: Create a function called process\_text that takes ownership of a String, modifies it, and returns it. In main, create a String variable, pass it to process\_text, then try to use it again. Observe how ownership transfer affects the variable’s usability and resolve it by returning ownership from the function.
3. **Borrowing with Multiple References**  
   Task: Write a program where a String variable greeting is borrowed immutably by two separate functions, each modifying it slightly but only printing their changes without altering the original String. Ensure both functions can access greeting simultaneously without mutable borrowing.
4. **Working with Arrays and Slices**  
   Task: Implement a function called slice\_average that takes a slice of integers (&[i32]) and returns the average as an f64. Call this function with an array and print the result. Handle cases where the slice might be empty by returning None.
5. **String Manipulation with Case Conversion**  
   Task: Create a function standardize\_case that takes a String, converts all characters to lowercase, and removes punctuation. For example, "Hello, WORLD!!" should become "hello world". Test this function with different input strings and print the results.
6. **Random Numbers with Conditional Filtering**  
   Task: Write a program that generates ten random numbers between 1 and 100, then filters out the even numbers, printing only the odd numbers. Use conditional logic and iteration to handle this filtering.
7. **Error Handling with Parsing**  
   Task: Write a program that prompts the user to input a series of floating-point numbers separated by commas. Parse the input into a vector of f64 numbers, handling any parsing errors gracefully. If parsing fails for any item, display an error message and skip that item.