To achieve the functionality where pressing Left Shift + Right Click shows a custom context menu instead of the normal one, you need to make some adjustments to your code. Here's a step-by-step guide on how to do it:

1. \*\*Detecting the Key and Mouse Click Combination\*\*: You've already implemented the logic to detect when the Left Shift key is pressed along with the Right Mouse button.

2. \*\*Showing the Custom Context Menu\*\*: When the desired key and mouse click combination is detected, you are correctly emitting a signal to show the context menu.

3. \*\*Constructing the Context Menu\*\*: You've designed the context menu and added actions to it. Make sure to correctly load your icons and that the paths are correct.

4. \*\*Handling Menu Actions\*\*: When an action is selected from the context menu, you need to define what each action does. You've included transformations like Uppercase, Lowercase, and Reverse, as well as Close and Exit actions.

5. \*\*Clipboard Interaction and Text Transformation\*\*: Your code for interacting with the clipboard and transforming text based on the selected action seems correct. Ensure that all logging and exception handling is done properly to avoid any silent failures.

6. \*\*Starting the Application and Listeners\*\*: You've set up the PyQt application, the main window, and started the keyboard and mouse listeners. This is necessary for capturing the key presses and mouse clicks.

Here are some additional considerations and potential improvements:

- \*\*Error Handling\*\*: Ensure that your code robustly handles any potential errors, especially when interacting with the clipboard or performing text transformations.

- \*\*Icon Paths\*\*: Verify that the paths to your icons are correct and that the icons are accessible.

- \*\*User Feedback\*\*: Consider providing user feedback for actions, such as displaying a message when a transformation is applied.

- \*\*Testing\*\*: Thoroughly test your application in different scenarios to ensure all functionalities work as expected.

- \*\*Code Optimization and Comments\*\*: Review your code for optimization opportunities and add comments for clarity, especially for complex sections.

- \*\*Compliance with OS Guidelines\*\*: Ensure that your application complies with the operating system's guidelines for keyboard and mouse interactions, particularly for unconventional uses like overriding default behaviors.

To apply adjustments to your code, consider the following enhancements and corrections:

1. \*\*Optimizing Key Detection\*\*:

- Your current implementation for detecting the left shift key press is good. However, consider handling cases where other keys might be pressed along with the shift key to avoid unintended behavior.

2. \*\*Enhancing Context Menu Creation\*\*:

- Ensure that the context menu is populated correctly with the desired actions. It seems you're adding actions both individually and within a loop. You might want to streamline this to avoid duplication or confusion.

- Verify that the icon paths are correctly specified and that the icons exist at those paths.

3. \*\*Improving Menu Action Handling\*\*:

- The `menu\_action\_selected` method handles different actions. Ensure that each action is properly defined and executed. For instance, the 'Exit' action correctly calls `sys.exit(0)`, but the 'Close' action might need more definition depending on what you intend it to do.

4. \*\*Clipboard Interaction and Text Transformation\*\*:

- The methods for interacting with the clipboard and transforming text seem appropriate. However, it's essential to handle exceptions properly to ensure the application does not crash when an error occurs during these operations.

5. \*\*Logging and Debugging\*\*:

- Make sure your logging statements are correctly formatted. For instance, in the `menu\_action\_selected` method, the logging statement `logging.info(f"Copied text:", copied\_text)` should be `logging.info(f"Copied text: {copied\_text}")`.

- Use logging strategically to help with debugging and understanding the flow of your application.

6. \*\*User Feedback\*\*:

- Consider providing visual or textual feedback to the user after performing actions, like showing a small popup message indicating the completion of a text transformation.

7. \*\*Code Clarity\*\*:

- Comment your code where necessary, especially in complex parts, to make it easier to understand and maintain.

8. \*\*Testing and Validation\*\*:

- Thoroughly test your application to ensure all features work as expected and handle edge cases, such as what happens if there is no text selected or the clipboard is empty.

With these adjustments, your application should be more robust, user-friendly, and maintainable.