Clean Code Development

Clean Code Development (CCD) focuses on writing code that is easy to read, understand, and maintain.

1. **Descriptive Variable and Function Names**
   - Names like self.welcome_label, self.converted_amount_field_label, and perform_conversion are more descriptive.
   - Descriptive names enhance code readability and make the purpose of variables and functions clear.

2. **Modularization**
   - The CurrencyConverterApp class has distinct methods for GUI setup, conversion, and history tracking.
   - Each method has a single responsibility, making the code more modular and easier to understand.

3. **Avoiding Global Variables**
   - The history variable is an attribute of the CurrencyConverterApp class, reducing its global scope.
   - Encapsulating related data within a class instance improves code organization and reduces the chance of unintended global interactions.

4. **Consistent Style**
   - Consistent font styles are maintained throughout the GUI, such as in labels and buttons. *self.welcome_label.config(font=('Courier', 15, 'bold'))*
   - *Consistency in style contributes to a visually cohesive and professional-looking user interface.*

5. **Error Handling**
   - Added error handling for the API request to handle exceptions and provide a more robust application.
   - Proper error handling ensures that the application can gracefully handle unexpected situations and provides useful information for debugging.

Clean Code Development Cheat Sheet

1. Single Responsibility Principle

Each method and class have a clear and single responsibility, promoting maintainability.

2. Meaningful Comments

Comments are used sparingly, providing explanations where the code's intent might not be immediately obvious.

3. Consistent Indentation

Follows consistent indentation style (4 spaces) for better readability.

4. Code Duplication

No significant code duplication is done, the functions are used to encapsulate repeated logic.

5. Avoid Magic Numbers

Numeric values are replaced with named constants or variables to improve code readability.

6. DRY Principle (Don't Repeat Yourself)

Reusable functions and methods are employed to avoid duplicating functionality.

7. Consistent Naming Conventions

Follows consistent naming conventions for variables, classes, and methods.

8. Explicit Imports

Explicitly imports modules and avoids wildcard imports for better code readability.

9. Avoid Deep Nesting

Limit the depth of nested structures, such as if statements and loops, to improve code readability.

10. Version Control Commit Messages

Commits are expected to be clear and represent small, meaningful changes.

11. Refactor Regularly

Code is structured to encourage easy refactoring as needed.