Clean Code Development

Clean Code Development emphasizes writing code that is not only functional but also easy to read, understand, and maintain. It follows a set of principles and practices that contribute to code clarity and maintainability. Now I will break down what's required for my project:

1. Meaningful Variable and Function Names:

- Variable names like **dob**, **user_name**, **questions**, and function names like **calculate_age**, **personality_quiz** are descriptive and convey their purpose.

2. Consistent Indentation and Formatting:

- The code follows consistent indentation and formatting throughout, making it easy to read.

3. Avoid Magic Numbers and Strings:

- Constants like **"YYYY-MM-DD"** and **QUESTION_COUNT** are used instead of magic numbers or strings.

4. Use of Comments for Clarification:

- Comments are used sparingly but effectively to explain the purpose of certain sections, such as user prompts and error messages.

5. Error Handling with Exception Handling:

- Exception handling is implemented when parsing the date of birth, providing clear error messages.

Clean Code Development Cheat Sheet

1. **Descriptive Naming:**
    - Use names that convey meaning and purpose.
2. **Single Responsibility Principle (SRP):**
    - Functions like **calculate_age** and **personality_quiz** have a single responsibility.
3. **Consistent Code Style:**
    - Maintain consistent formatting and indentation.
4. **Avoid Duplication (DRY Principle):**
    - The code structure avoids unnecessary duplication.
5. **Limit Function/Method Length:**
    - Functions are relatively short and focused.
6. **Avoid Deep Nesting:**
    - The code minimizes nesting, enhancing readability.
7. **Use Version Control Effectively:**
    - Although not visible in the code, effective use of version control is encouraged.
8. **Document Code When Necessary:**
    - The code is mostly self-explanatory; however, additional comments can be added where needed.
9. **Test-Driven Development (TDD):**
    - While not explicitly evident in the code, adding tests for critical functionalities is recommended.
10. **Minimize Global Variables:**
    - The code doesn't rely heavily on global variables.
11. **Readable Code over Clever Code:**
    - Prioritize readability over overly clever solutions.

12. Code Reviews:

- Periodic code reviews can help identify areas for improvement.

13. Continuous Refactoring:

- Periodically revisit the code to improve design and maintainability.

14. Use of Meaningful Constants:

- Constants like **"YYYY-MM-DD"** enhance code readability.

15. Error Handling:

- Effective use of try-except blocks for error handling.