FUNCTIONAL PROGRAMMING

1. Final Data Structures:

- The `quiz_data` dictionary serves as a definitive and immutable data structure, embodying the principles of functional programming. Its static nature ensures that the set of quiz questions and answers remains constant throughout the program's execution, contributing to predictability and ease of reasoning about program behavior.

2. Side-Effect-Free Functions:

- The `run_quiz` function stands as a testament to the principles of functional programming, steering clear of side effects and external state modifications. By isolating its computations and interactions within well-defined boundaries, it establishes a robust and reliable foundation that aligns with the functional paradigm.

3. Use of Higher-Order Functions:

- The application of higher-order functions is evident in the testing infrastructure, where the `patch` function facilitates the modification of the `input` function's behavior. This utilization of higher-order functions enhances the testability of the codebase, allowing for isolated and controlled testing scenarios.

4. Functions as Parameters and Return Values:

- The `run_quiz` function dynamically incorporates functions as both parameters and return values, showcasing a functional approach to handling dynamic input and output. This flexibility not

only facilitates the adaptability of the function but also fosters composability, a hallmark of functional programming.

5. Use of Closures / Anonymous Functions:

- Although explicit closures or anonymous functions are not prevalent, the encapsulation of variables within the `run_quiz` function signifies a form of lexical closure. This deliberate encapsulation of state within the function's scope enhances readability and encapsulates contextual information where needed.

6. Immutable State Management:

- The code emphasizes immutability by employing constant data structures and avoiding mutable state changes within functions. This design choice minimizes unexpected side effects, contributes to code predictability, and aligns with the functional programming paradigm's emphasis on immutability.

7. Declarative Style:

- The structure of the code leans towards a declarative style, where the focus is on expressing what the program should accomplish rather than prescribing step-by-step procedures. This approach aligns with functional programming principles, emphasizing clarity and conciseness.

8. Data Transformation Pipelines:

- While the specific code does not include complex data transformation pipelines, the design facilitates the potential extension of such pipelines. This aligns with functional

programming principles, where operations are modeled as transformations on immutable data structures.

In summary, the codebase showcases a robust integration of functional programming principles, including immutable data structures, side-effect-free functions, and the judicious use of higher-order functions. These elements collectively contribute to code reliability, maintainability, and adherence to the functional paradigm.