1. **Match Baud Rates on Both Ends**

   Always ensure that both transmitting and receiving devices operate at the same baud rate (e.g., 9600, 115200 bps) to avoid data corruption.

2. **Properly Configure GPIO Pins**

   Configure the microcontroller's TX and RX pins in alternate function mode and ensure correct electrical characteristics (push-pull/open-drain, pull-up/down).

3. **Enable Peripheral Clocks**

   Don't forget to enable the UART or USART peripheral clock in your microcontroller's RCC or clock control module.

4. **Set Frame Format Correctly**

   Choose correct frame parameters like word length (usually 8 or 9 bits), number of stop bits (1 or 2), and parity (none, even, odd) to match the connected device.

5. **Use Pull-up Resistors on RX Line (if needed)**

   On some boards, the RX line may float and cause false starts. Use a pull-up resistor if required.

6. **Implement Interrupt-Based Communication**

   For efficiency, use UART receive/transmit interrupts instead of polling. This avoids CPU blocking during data transmission.

7. **Use DMA for Large Data Transfers**

   In high-speed applications, Direct Memory Access (DMA) can offload CPU

and improve UART throughput.

8. **Design Circular Buffers for RX and TX**

   Implement ring buffers for storing incoming and outgoing data streams. This prevents data loss and makes buffering more robust.

9. **Debounce or Delay After Power-Up**

   Some UART peripherals need a few clock cycles or milliseconds of delay after reset or power-up before being configured.

10. **Verify Clock Accuracy**

    UART timing depends on system clock. Ensure the main clock is accurate enough for the selected baud rate.

11. **Avoid Using printf in ISR**

    Don't use blocking functions like printf inside interrupt handlers. It can cause system hangs and missed UART events.

12. **Handle UART Errors Gracefully**

    Monitor and clear error flags such as framing error, overrun error, and noise error in status registers.

13. **Check TX Buffer Empty Before Sending**

    Always check that the transmit buffer is empty before writing new data, especially when sending manually without interrupts.

14. **Include Start and Stop Bits in Framing**

    Remember that UART automatically adds start and stop bits. You don't need

to add them manually in software.

15. **Use Hardware Flow Control if Required**
    For reliable high-speed communication, enable RTS/CTS hardware flow control if supported by both devices.

16. **Watch Out for Crosstalk and Noise**
    Use shielded cables and proper grounding if UART lines run over long distances or noisy environments.

17. **Don't Forget NVIC Configuration (for ARM MCUs)**
    When using interrupts on ARM-based MCUs, make sure to enable the UART IRQ in the Nested Vectored Interrupt Controller (NVIC).

18. **Use External Level Shifters (if voltage mismatch)**
    If UART devices operate at different voltage levels (e.g., 5V and 3.3V), use a level shifter to prevent damage.

19. **Verify Data Using Serial Terminals**
    Tools like Tera Term, PuTTY, or RealTerm help visualize sent/received UART data and debug issues.

20. **Document Your UART Settings**
    Clearly document UART configurations (baud rate, parity, stop bits, etc.) in your code or header files to ensure compatibility in future development.