

Leistungskurs Praktische Q3 Hessen

Skript

Shamsher Singh Kalsi

Berufliches Gymnasium — Ferdinand-Braun Schule
Kursleiter: Herr Sebastian Stolz

12. September 2025

Inhaltsverzeichnis

1	Einführung	2
----------	-------------------	----------

1 Einführung

05.09.2025

Aufgabe 1.1: Serielle Kommunikation

1. Beantworte folgende Fragen schriftlich:

- An welcher Stelle spielt die serielle Kommunikation heutzutage eine Rolle?
- Erkläre die Begriffe: Startbit, Datenbit, Stoppbit.
- Ein PC sendet den Buchstaben 'A' mit 1 Startbit, 8 Datenbits, 1 Stoppbit. Skizziere das resultierende Bitmuster
- Welche Parameter müssen Sender und Empfänger bei RS232 vorab gemeinsam einstellen?
- Warum reicht ein einziger Draht für die Übertragung?

2. Aufgabe 2 – Serielle Kommunikation (RS232) in Java mit Hilfe eines Emulators

- Warum brauchen Sender und Empfänger die gleiche Baudrate?
- Was passiert, wenn der Sender schneller schreibt als der Empfänger lesen kann? Welche Lösungen gibt es für dieses Problem?

Lösung 1.1: Serielle Kommunikation

1. Beantworte folgende Fragen schriftlich:

- Serielle Kommunikation wird heute noch in eingebetteten Systemen, Industrieanlagen, Messgeräten und auch beim Serverzugriff (Konsolenport) genutzt, da sie einfach, robust und für kurze Distanzen ausreichend ist.
- **Startbit:** signalisiert Beginn eines Zeichens (logisch 0) und dient zur Synchronisation. **Datenbits:** eigentliche Nutzinformation, meist 8 Bit, LSB zuerst. **Stopbit:** beendet die Übertragung (logisch 1), Leitung geht in Idle-Zustand.
- Beispiel: Der Buchstabe A hat den ASCII-Wert $0x41 = 01000001$. Übertragung (LSB zuerst) mit 1 Startbit und 1 Stopbit:

$\underbrace{1}_{\text{Idle}} \quad \underbrace{0}_{\text{Start}} \quad 10000010 \quad \underbrace{1}_{\text{Stop}} \quad 1$

- Sender und Empfänger müssen sich bei RS232 vorab einigen auf: Baudrate, Datenbits, Parität, Stopbits, sowie ggf. Art der Flusskontrolle.
- Ein einzelner Draht genügt pro Richtung, weil Bits nacheinander mit fester Baudrate gesendet werden; Start- und Stopbits übernehmen die Synchronisation. Für echte Vollduplex-Kommunikation sind jedoch zwei Leitungen (Tx/Rx) plus Masse üblich.

2. Aufgabe 2 – Serielle Kommunikation (RS232) in Java mit Hilfe eines Emulators

- Beide Seiten brauchen dieselbe Baudrate, da es kein separates Taktsignal gibt. Unterschiedliche Baudraten führen zu falscher Bitinterpretation.
- Wenn der Sender schneller schreibt als der Empfänger liest, läuft dessen Puffer über und Daten gehen verloren. Lösungen: Hardware-Flowcontrol (RTS/CTS), Software-Flowcontrol (XON/XOFF), größere FIFO-Puffer oder Protokolle mit Bestätigung (ACK/NACK).

```

1  import com.fazecast.jSerialComm.SerialPort;
2  public class Sender {
3      public static void main(String[] args) throws Exception {
4          SerialPort sp = SerialPort.getCommPort("COM5");
5          sp.setBaudRate(9600);
6          sp.openPort();
7          sp.getOutputStream().write("Hallo, COM6\n".getBytes());
8          sp.closePort();
9      }
10 }
```

1	0	1	0	0	0	0	0	1	0	1	1
Idle		Start		Datenbits						Stop	

09.09.2025

Aufgabe 1.2: Steuerung eines Mikroprozessors mit der seriellen Schnittstelle

1. Schreibe ein (Python-)Programm, welches einen selbst gewählten Sensor auf einem Raspberry-Pi oder einem Arduino steuert.
2. Erweitere das Programm so, dass es über eine serielle Schnittstelle angesprochen werden kann. Emuliere die serielle Schnittstelle mit Hilfe von Software oder nutze einen RS232/TTL Wandler mit MAX3232
3. Nutze die Klasse SSerial aus dem Moodle-Kurs, um von einem Laptop oder PC mit einem Java-Programm über die serielle Schnittstelle den Sensor zu steuern.

Lösung 1.2: Steuerung eines Mikroprozessors mit der seriellen Schnittstelle

1. Schreibe ein Python-Programm, das einen Sensor bzw. ein Aktor-Device auf einem Raspberry Pi steuert.

Listing 1: Raspberry Pi: LED-Steuerung + serielle Steuerung (pySerial + gpiozero)

```
1      # requirements: gpiozero, pyserial
2      # Ein Python-Skript, das eine LED steuert und auf Befehle über die serielle
      Schnittstelle hört.
3      from gpiozero import LED
4      import serial
5      import time
6
7      # Pin-Nummer der LED und Details zur seriellen Schnittstelle.
8      LED_PIN = 17
9      SERIAL_PORT = '/dev/ttyUSB0'
10     BAUD = 9600
11
12     led = LED(LED_PIN)
13     ser = serial.Serial(SERIAL_PORT, BAUD, timeout=1)
14
15     # Verarbeitet die empfangenen Befehle.
16     def handle_line(line):
17         line = line.strip().upper()
18         if line == 'LED ON':
19             led.on()
20             ser.write(b'OK\n')
21         elif line == 'LED OFF':
22             led.off()
23             ser.write(b'OK\n')
24         elif line == 'STATUS':
25             ser.write(b'ON\n' if led.is_lit else b'OFF\n')
26         else:
27             ser.write(b'ERR Unknown command\n')
28
29     # Hauptschleife zum Empfangen von Daten.
```

```

30     try:
31     while True:
32         raw = ser.readline()
33         if raw:
34             handle_line(raw.decode('utf-8'))
35             time.sleep(0.01)
36         finally:
37             ser.close()
38             led.off()

```

2. Ein Arduino-Beispiel, das serielle Kommandos entgegennimmt und einen digitalen Pin steuert:

Listing 2: Arduino: Serial command handler

```

1 // Ein Arduino-Sketch, der auf serielle Kommandos reagiert.
2 const int LED_PIN = 13;
3
4 void setup() {
5     pinMode(LED_PIN, OUTPUT);
6     Serial.begin(9600); // Startet die serielle Kommunikation.
7 }
8
9 void loop() {
10     if (Serial.available() > 0) { // Prüft, ob Daten verfügbar sind.
11         String cmd = Serial.readStringUntil('\n'); // Liest eine Zeile.
12         cmd.trim();
13         if (cmd == "LED ON") {
14             digitalWrite(LED_PIN, HIGH);
15             Serial.println("OK");
16         } else if (cmd == "LED OFF") {
17             digitalWrite(LED_PIN, LOW);
18             Serial.println("OK");
19         } else if (cmd == "STATUS") {
20             Serial.println(digitalRead(LED_PIN) ? "ON" : "OFF");
21         } else {
22             Serial.println("ERR");
23         }
24     }
25 }

```

Die Arduino-API stellt 'Serial.available()' und 'Serial.read()' / 'readStringUntil()' bereit; 'available()' gibt die Anzahl bereits empfangener Bytes an, 'read()' liefert das nächste Byte oder -1, wenn nichts da ist — das ist der übliche Pattern für nicht-blockierende Abfragen auf Arduino.

3. Java-Client mit der Klasse Serial (wie in deinem Moodle-Skript beschrieben). Das Beispiel öffnet den Port, schickt einen String, liest eine Antwortzeile und parst eine Zahl mit Double.parseDouble(...):

Listing 3: Java: Steuerprogramm (Nutzungsbeispiel der in der Aufgabenstellung beschriebenen Serial-Klasse)

```
1 // Ein Java-Client zum Steuern des Mikroprozessors.
2 public class SerialController {
3
4     public static void main(String[] args) {
5         Serial s = new Serial("COM5", 9600, 8, 1, 0);
6
7         if (!s.open()) {
8             System.err.println("Port konnte nicht geöffnet werden");
9             return;
10        }
11
12        // Sendet einen Befehl und liest die Antwort.
13        s.write("STATUS\n");
14        String reply = s.readLine();
15        System.out.println("Reply: " + reply);
16
17        // Versucht, die Antwort in eine Zahl umzuwandeln.
18        try {
19            double val = Double.parseDouble(reply.trim());
20            System.out.println("Parsed value: " + val);
21        } catch (NumberFormatException e) {
22            System.out.println("Keine Zahl: " + reply);
23        }
24
25        s.close();
26    }
27 }
```


Theorem 1.1: Fundamentalener Satz

Inhalt des Theorems ...

Beispiel 1.1: Erstes Beispiel

Dieses Beispiel illustriert den Satz.

Aufgabe 1.3: Rechenaufgabe

Bearbeite folgende Aufgabe ...

Lösung 1.3: zur Aufgabe

Hier die Lösungsschritte ...

Hinweis

Ein kurzer Hinweis.

Terminal: Beispielcode

```
echo "Hallo Welt"ls -la
```