Informatikwerkstatt, Foliensatz 5
Test mit Python

G. Kemnitz

Institut für Informatik, TU Clausthal (IW-F5)

15. November 2022

Inhalt:

Interaktive Übungen:

1. Test mit Python-Scripten (python*).
2. Modularisierung und Modultest (com_pc)

Wiederholung

W5.1: Kopieren einer Zeichenfolge

Der Inhalt von Feld »a« soll in Feld »b« kopiert werden.

```
//Vereinbarung der Felder a und b
                                    ;
//Zeigern pa und pb auf a[0] bzw. b[0]
                                    ;
uint8_t idx; //Schleifenzähler
int main(){
 // Wiederhole 5 mal
 for(                          ){
  . . .              ;//Inhalt von pa
  . . .              ;//nach pb kopieren
  . . .              ;//pa um eins erhöhen
  . . .              ;//pb um eins erhöhen
 }
}
```

Var.	Adr.	AW
a[0]	200	'T'(54)
a[1]	201	'e'(65)
a[2]	202	'x'(78)
a[3]	203	't'(74)
a[4]	204	0
b[0]	205	0
b[1]	206	0
b[2]	207	0
b[3]	208	0
b[4]	209	0
b[5]	20A	0
pa.1	20B	2
pa.0	20C	0
pb.1	20D	2
pb.0	20E	5
idx	20F	0

W5.2: Kopieren bis zur Abschluss-Null

Vervollständigen Sie die Beschreibung der Funktion »strcpy« zum Kopieren der Zeichenkette von Quelladresse »src« zur Zieladresse »dest« bis zur abschließenden Null einschließlich dieser

```
void strcpy(uint8_t *desc,
            const uint8_t *src){
 while(                    ){
  . . .          ;//Inhalt von src
  . . .          ;//nach dest kopieren
  . . .          ;//scr um eins erhöhen
  . . .          ;//dest um eins erhöhen
 }
}

strcpy(b, a);    //Testbeispiel
```

Var.	Adr.	Inh.
a[0]	200	'T'(54)
a[1]	201	'e'(65)
a[2]	202	'x'(78)
a[3]	203	't'(74)
a[4]	204	0
b[0]	205	0
b[1]	206	0
b[2]	207	0
b[3]	208	0
b[4]	209	0
b[5]	20A	0
pa.1	20B	2
pa.0	20C	0
pb.1	20D	2
pb.0	20E	5
idx	20F	0

W5.3: Modultest

1. Wie wurden auf Foliensatz IW-F3 Testbeispiele und Testsätze programmiert?
2. Vereinbaren Sie einen Verbund »tb« für die Beschreibung eines Testbeispiels mit zwei »uint8_t« Eingabewerten »a« und »b« und einem »int16_t« Ausgabewert »y« sowie einen Testsatz als Feld für die Testbeispiele in der nachfolgenden Tabelle:

Testbeispiel	a	b	c
0	34	17	-2654
1	2	200	10025
2	78	4	-7546

3. Welche Aufgaben hat ein Testrahmen und was für einen Programmablauf hatten die bisher besprochenen Testrahmen?

Lösung

1. Testbeispiele wurden als Verbund von Eingabe- und Sollausgabewerten und Testsätze als initialisierte Felder von Testbeispielen beschrieben.
2. Verbund für die Beschreibung eines Testbeispiels mit zwei »uint8_t« Eingabewerten und einem »int16_t« Ausgabewert:

   ```
   struc tb {uint8_t a,b; int16_t y;}
   ```

   Initialisiertes Feld mit den vorgegebenen Testbeispielen:

   ```
   struc tb Testsatz[]={{ 34, 17,-2654},
           {2,200,10025},{ 78,  4,-7546}};
   ```

3. Der Testrahmen ist ein Hauptprogramm, das das Testobjekt in einer Schleife »wiederhole für alle Testbeispiele« abarbeitet, die Ergebnisse kontrolliert, Fehler zählt, ...

Test mit Python

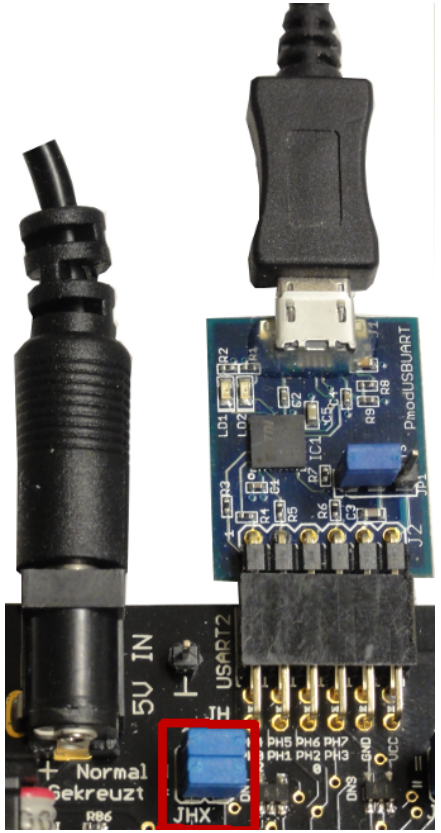
Python als Programmiersprache für Tests

Python: Interpretersprache vorzugsweise zum Experimentieren und für Tests, einfach zu erlernen, gut zu debuggen, ... Installation für zu Hause und Kurzeinführung siehe später Zusatzteil.

Experimente:

1. Test des Echo-Programms mit einem Python-Script.
2. Untersuchung der Übertragungsdauer.
3. Modultest mit Python-Skript auf dem PC.

Wiederholung: Test mit HTerm



Hardware vorbereiten:

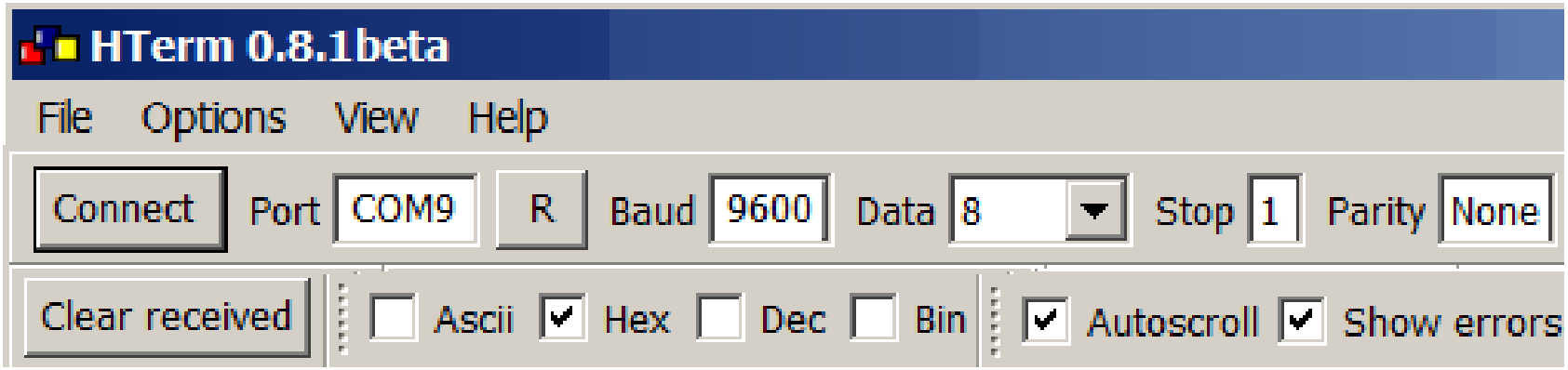
- Spannung ausschalten.
- Jumper JHX »gekreuzt (=)«.
- PModUSBUSART Kontrolle, Jumper wie im Bild, und und an JH oben stecken.
- PModUSBUSART mit PC verbinden. Spannung einschalten.

Software vorbereiten:

- Projekt Echo öffnen.
- »Dragon« und Compileroptimierung »-O0« auswählen.
- Übersetzen und im Debugger laufen lassen.

Verbindung mit HTerm herstellen

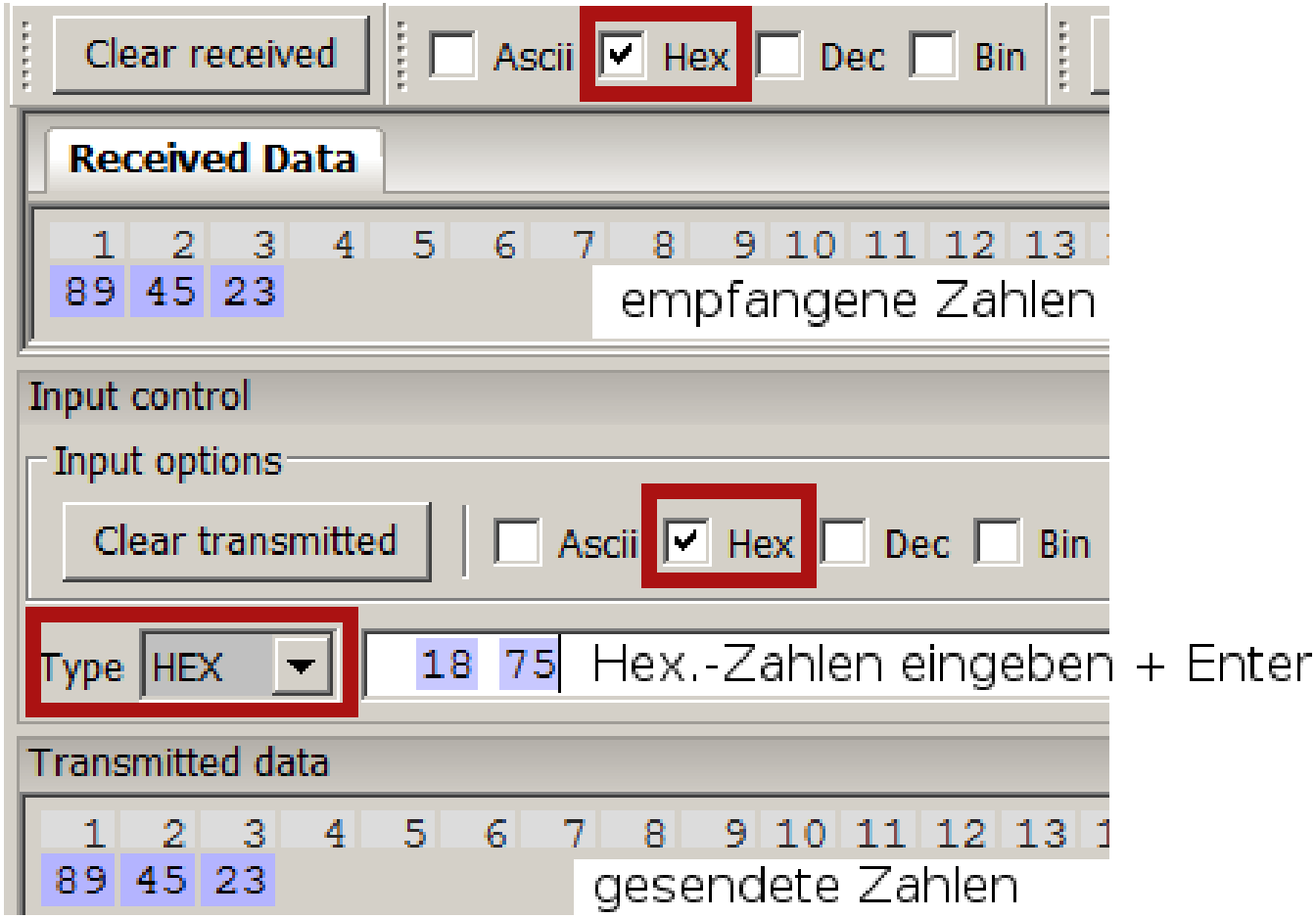
- Auf dem PC HTerm starten.
- COM-Port auswählen[1].
- 9600 Baud, 8 Daten-, 1 Stopp- und kein Paritätsbit einstellen.
- Verbindung herstellen (Connect).



[1] Die COM-Schnittstelle, die nach Anstecken des USB-Kabels vom PmodUSBUART und »R« (Refresh Comport List) als neuer Port erscheint.

2. Test mit Python

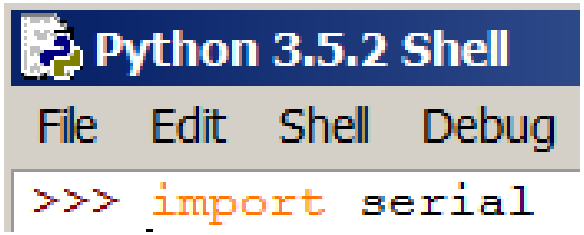
Für die Eingabe »HEX« auswählen. Für die Darstellung der Sende- und Empfangsdaten nur bei »Hex« $\surd$ setzen.



- Alle versendeten Zahlen werden zurückgesendet.

Test des Echoprogramms mit Python

- Verbindung mit HTerm schließen (Disconnect)
- Start der Programmierkonsole »Idle« von
  `> Start > Python 3.5 > IDLE (Python 3.5 ...)`
- Import des Moduls für die serielle Kommunikation:



2. Test mit Python

- Kommunikationsverbindung öffnen[2]:

```
ser = serial.Serial("COM9")
```

- Zeichenfolge an Variable zuweisen:

```
send = "Hallo Welt!"
```

- Anschauen von Typ, Wert und Länge der Zeichenkette:

`>>> type(send)` `<type 'str'>`	Funktion zur Bestimmung des Datentyps. Der Datentyp ist "Zeichenkette (string)".
`>>> send` `'Hallo Welt!'`	Bei Eingabe des Variablennamens wird der Wert angezeigt.
`>>> len(send)` `11`	Die Funktion "len()" liefert die Anzahl der Elemente der Zeichenkette.

[2] Denselben COM-Port wie im HTerm benutzen. Das Programm P05\Python\list_com_ports.py listet alle COM-Ports, die sich öffnen lassen. 8N1, 9600 Baud ist der Standardwert und muss deshalb nicht eingestellt werden.

2. Test mit Python

- Zeichenkette senden:

```
ser.write(send.encode("ascii"))
```

- Auf 11 Zeichen warten und diese lesen:

```
y = ser.read(11)
```

- Anzeige von Typ, Wert und Länge der empfangenen Daten mit print(<Zeichenkette>):

```
print("type(y)␣␣␣␣␣␣␣␣␣:␣" + str(type(y)))
a = y.decode()
print(a, type(a)); print(y, type(y))
print("Empfangene␣Daten:␣" + a)
print("len(a)␣␣␣␣␣␣␣␣␣␣:␣%i" %len(a))
```

---

a + b	Verkettung der Zeichenketten a und b.
str(x)	Konvertierung von »x« in eine Textdarstellung.
"... %i"%(<w>)	Formatierte Ausgabe des Wertes »w«.

Zusammenfassen zum Programm »scom.py«

```
import serial
ser = serial.Serial("COM9")
send = "Hallo␣Welt!".encode("ascii")
ser.write(send)
y = ser.read(len(send))
print("Empfangene␣Daten:␣" + y.decode())
ser.close()
```

- encode/decode: Umwandlung Zeichenkette $\Leftrightarrow$ Byte-Vektor.
- Zeilen eines Blocks **müssen** gleiche Einrücktiefe haben!

Programmdatei in der »Idle« öffnen:

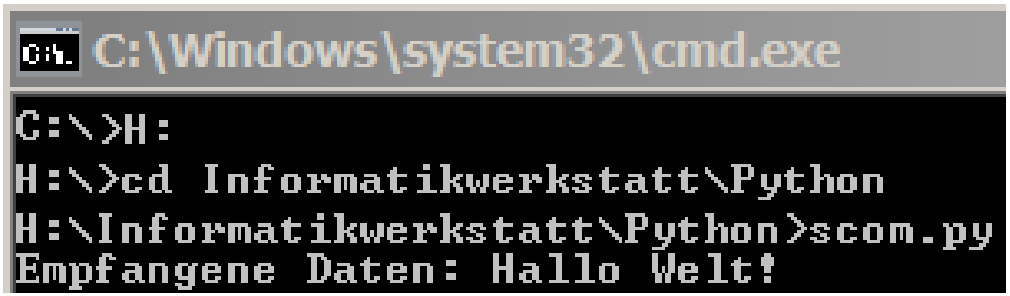
`File > Open > H:~\Informatikwerkstatt\P05\Python\scom.py`

Programmstart mit Ausgabe auf der »Idle«:

`Run > Run Module (F5)`

Start auf der Konsole

- Konsole öffnen (»Windows-Taste + R«, im sich öffnenden Feld »cmd« eingeben, Enter).
- Wechsel in das Verzeichnis mit dem Python-Programm, im Bild »H:\Informatikwerkstatt\Python«. Eingabe Programmname + Enter:



- Die Programmausgabe »Empfangene Daten: Hallo Welt!« erfolgt auf der Konsole.

Messung der Übertragungsdauer mit »scom_t.py«

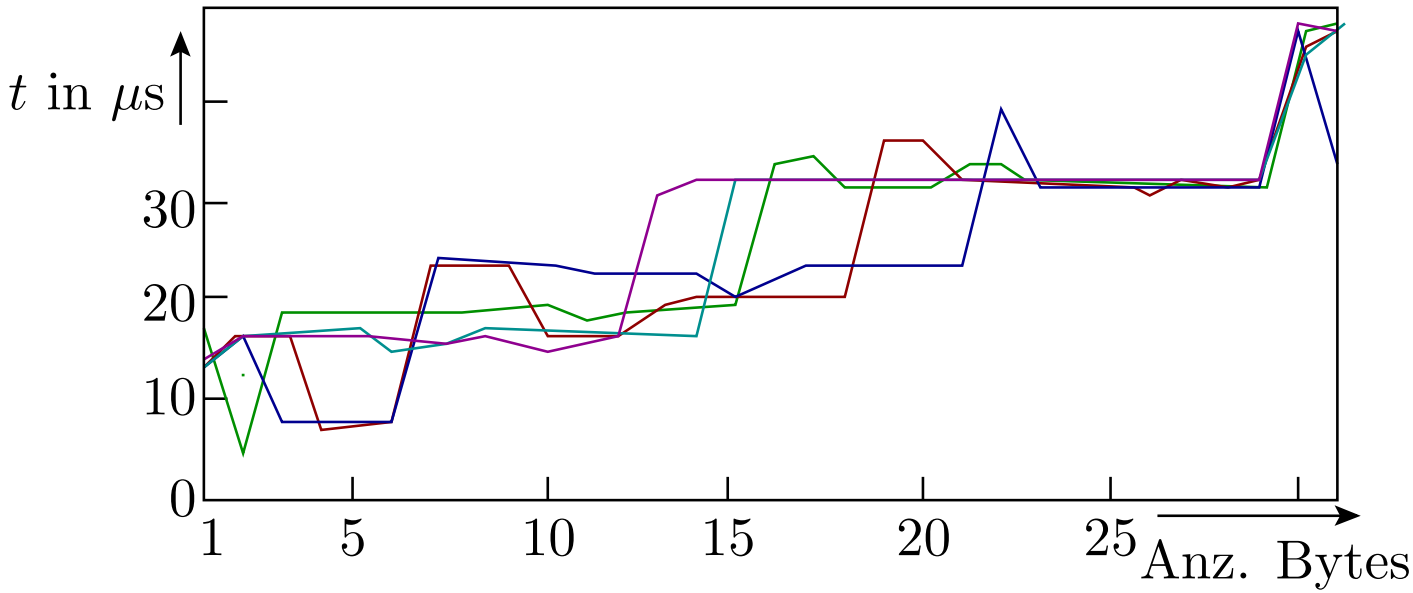
```
import serial
# Funktion clock() aus Modul "time"
from time import clock
ser = serial.Serial("COM9")
send = "Hallo␣Welt!".encode("ascii")
ts = clock();      #Startzeit in Sekunden nach ...
ser.write(send)
y = ser.read(len(send)).decode()
dt = clock()-ts    #Zeitdifferenz zur Startzeit
print("Empfangene␣Daten:"+y
          +"␣dt=␣"+str(dt*1E3)+"␣ms")
ser.close()
```

Programmstart und Ausgabe auf der Konsole:

```
H:\Informatikwerkstatt\Progr_IW\Python>scom_t.py
Empfangene Daten: Hallo Welt! dt= 13.6730654185 ms
```

Übertragungszeit und Paketgröße

Die Übertragung wird über USB und später auch über Bluetooth getunnelt und erfolgt in Paketen aus mehreren Bytes. Dauer abhängig von der Paketgröße und nicht deterministisch.



Das nachfolgende Programm scom_txy.py bestimmt für Bytefolgen der Länge 1 bis 31 die Übertragungsdauer und scom_txy5.py wiederholt das 5-mal.

Zeitmessung für Paketgröße 1 bis 30 »scom_txy.py«

```
import serial
from time import clock
ser = serial.Serial("COM9", timeout=1)
send = "Das␣ist␣ein␣sehr␣langer␣String!".encode("ascii")
l = 1; dt_list=[];   #leere Liste fuer dt-Werte
while l<=len(send):  #Wiederhole bis Gesamtlaenge
  ts = clock();      #neuer Block => einruecken
  ser.write(send[:l])#Sende die ersten l Zeichen
  y = ser.read(l).decode()#Warte auf l Zeichen
  dt = clock()-ts    #Zeitdifferenz zur Startzeit
  dt_list.append(dt) #Differenzzeiten an Liste haengen
  print("Empf.␣Daten:␣"+y, "dt=␣"
                       +str(dt*1000)+"␣ms")
  l = l+1;           #Ende des Schleifenkoerpers
ser.close()          #danach Einruecktiefe ruecksetzen
```

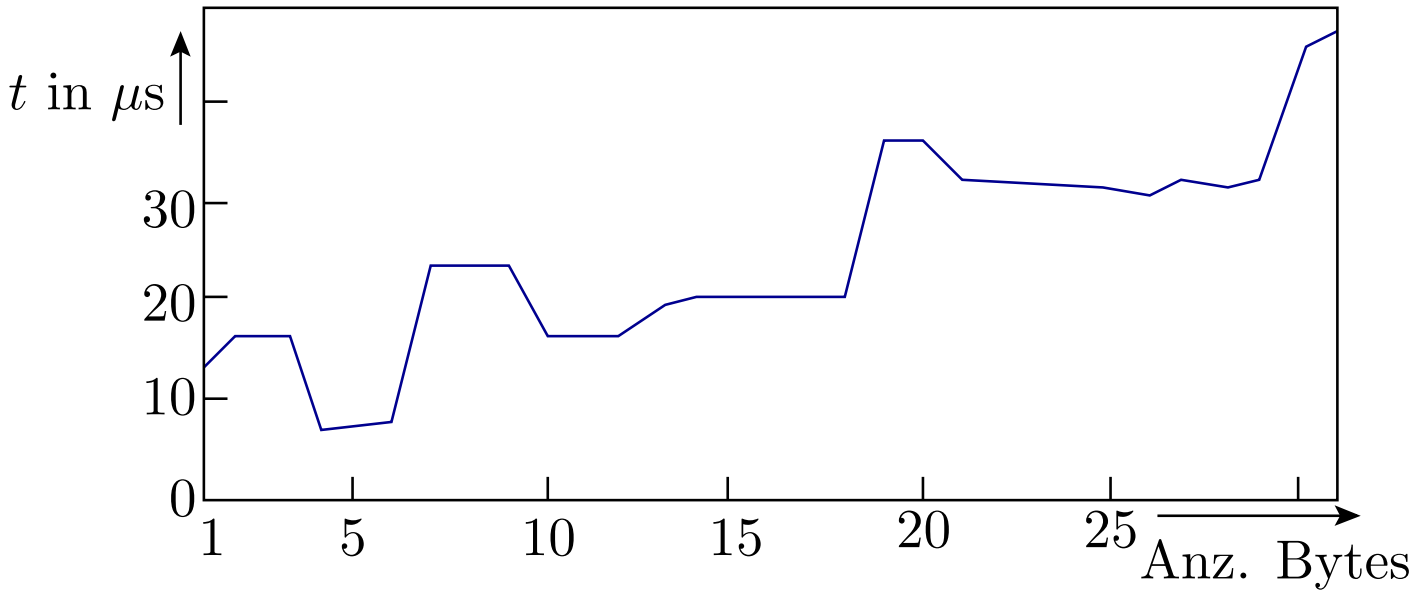
- Anweisungen für graphische Ausgabe siehe übernächste Folie.

Test von »scom_txy.py« auf der Konsole

```
H:\Informatikwerkstatt\Progr_IW\Python>scom_txy.py
Empfangene Daten: D dt= 13.2549278489 ms
Empfangene Daten: Da dt= 14.921187855 ms
Empfangene Daten: Das dt= 14.40439923316 ms
Empfangene Daten: Das  dt= 14.4265738178 ms
Empfangene Daten: Das i dt= 14.3878389836 ms
Empfangene Daten: Das is dt= 14.2295889771 ms
Empfangene Daten: Das ist dt= 14.368306033 ms
Empfangene Daten: Das ist  dt= 14.3656574973 ms
Empfangene Daten: Das ist e dt= 14.2249540396 ms
Empfangene Daten: Das ist ei dt= 14.1339106258 ms
Empfangene Daten: Das ist ein dt= 14.1051078004 ms
Empfangene Daten: Das ist ein  dt= 14.4951046784 ms
Empfangene Daten: Das ist ein s dt= 30.4876252137 ms
Empfangene Daten: Das ist ein se dt= 29.8907114861 ms
Empfangene Daten: Das ist ein seh dt= 30.3879740589 ms
Empfangene Daten: Das ist ein sehr dt= 30.0079091897 ms
Empfangene Daten: Das ist ein sehr  dt= 30.3273888054 ms
Empfangene Daten: Das ist ein sehr l dt= 30.1208030228 ms
Empfangene Daten: Das ist ein sehr la dt= 30.1522543839 ms
Empfangene Daten: Das ist ein sehr lan dt= 30.1446398438 ms
Empfangene Daten: Das ist ein sehr lang dt= 30.2389939272 ms
Empfangene Daten: Das ist ein sehr lange dt= 30.3604955013 ms
```

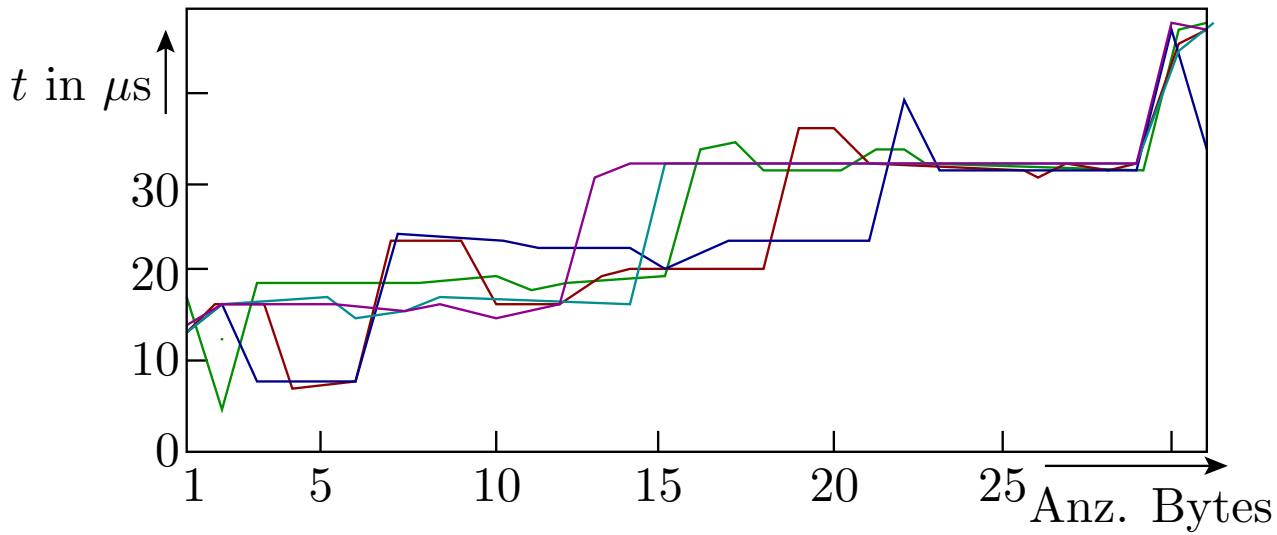
Graphische Ausgabe am Ende von »scom_txy.py«

```
# Import der Klasse "pyplot"
import matplotlib.pyplot as plt
plt.plot(range(1, len(dt_list)+1), dt_list)
plt.xlabel("Anzahl der Bytes") #plot erzeugen
plt.ylabel("dt in ms")         #Achsenbeschrift.
plt.show()                     #anzeigen
```



5-fache Wiederholung (Programm: scom_txy5.py)

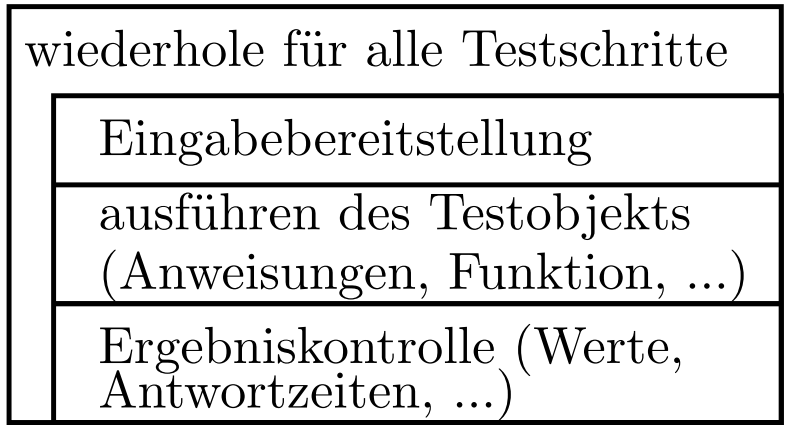
```
...; plt.hold(True)
for idx in range(5):
 <Bestimme Übertragungsdauer für 1 bis 31 Byte>
 plt.plot(range(1,len(dt_list)+1), dt_list)
ser.close()
plt.show(); ...
```



- Bei 9600 Bitzeiten pro s, 1 Startbit + 8 Datenbits + 1 Stoppbit dauert eine Byteübertragung $\geq$1 ms.
- Für 21 bis 28 Byte große Pakete werden etwa 30 ms benötigt, d.h. fast max. Durchsatz.
- Auf anderen Rechnern, zeitgleichen Übertragungen über USB, Nutzung von Bluetooth, ... kann die Übertragung auch so lange dauern, dass es stört.
- Falls ihre Zielanwendung später wegen zu großen Übertragungszeiten nicht funktioniert, kann das Übertragungsverhalten in der dargestellten Weise untersucht und so zielgerichtet nach alternativen Lösungen gesucht werden.

Modultest vom PC aus

Testrahmen



Die Basisfunktionen für den Test vom PC:

- Übergabe von Eingabe-Bytes und die
- Rückgabe von Ergebnis-Bytes zur Auswertung.

sind im Echo-Programm enthalten.

Modularisierung des Echoprogramms

Aufteilung des Echoprogramms »echo.c« vom vorhergen Foliensatz in nachnutzbare Module für den Test von Programmbeisteinen:

```
int main(void){
// ---------- Initialisierung ---------------------
 UCSR2C=0b110;                    // Format 8N1
 UBRR2=51;                        // 9600 Baud
 UCSR2B=(1<<RXEN2)|(1<<TXEN2);// Empf. + Sender ein
// ------------------------------------------------
 while(1){
// ---------- Empfangen eines Bytes --------------
  while (!(UCSR2A & (1<<RXC2)));//warte auf Byte
  daten = UDR2;                   //Byte übernehmen
// ---------- Versenden eines Bytes --------------
  while (!(UCSR2A &(1<<UDRE2)));//warte Puffer frei
  UDR2 = daten;                   //Byte übergeben
 } // ------------------------------------------------
}
```

Funktionen für die PC-Kommunikation (com_pc.c)

```
#include <avr/io.h>
//Initialisierung von USART2 (8N1, 9600 Baud)
void com_pc_init(){
 UCSR2C=0b110;                    //Format 8N1
 UBRR2=51;                        //9600 Baud
 UCSR2B=(1<<RXEN2)|(1<<TXEN2); //E+S ein
}
//ein Byte empfangen
uint8_t getByte(){
 while (!(UCSR2A & (1<<RXC2)));//warte auf ein Byte
 return UDR2;                     //Byte zurueckgeben
}
//ein Byte versenden
void sendByte(uint8_t dat){
 while (!(UCSR2A & (1<<UDRE2)));//warte Puffer frei
 UDR2 = dat;                       //Byte uebernehmen
}
```

Header »com_pc.h« für den Export

```
#ifndef COM_PC_H_
#define COM_PC_H_

#include <avr/io.h>

 void com_pc_init();           //Init. USART2
 uint8_t getByte();            //Byte empfangen
 void sendByte(uint8_t dat); //Byte versenden

#endif /* COM_PC_H_ */
```

- Der Header wird in die C-Dateien, die die Funktionen definieren und nutzen eingefügt.
- In nutzenden Dateien macht das die Aufrufschnittstelle bekannt.
- In der definierenden Datei dient das zur Kontrolle, dass Definition und Implementierung der Aufrufschnittstellen gleich sind.
- #ifndef ... #define ... #endif – Precompiler-Anweisungen zur Verhinderung einer Mehrfacheinbindung.

Echoprogramm aus Funktionsbausteinen

```
#include <avr/io.h>    // Anmerkung *1
#include "com_pc.h"
uint8_t d;
int main(void){
 com_pc_init();        //Init. USART2
 while(1){             //Endlosschleife
  d = getByte();       //Byte empfangen
  sendByte(d);         //Byte zurücksenden
 }
}
```

*1: »avr/io.h« ist bereits in »com_pc.h« eingefügt. Ohne dem »Precompiler-Konstrukt« auf Folie zuvor Mehrfacheinfügung.

Modultest vom PC – Ein Testobjekt

Testobjekt sei folgende Berechnungssequenz:

```
uint8_t a, b, s, d, q, r;
uint16_t p;
...
s = a + b;      // Summe
d = a - b;      // Differenz
p = a * b;      // Produkt
q = a/b;        // ganzzahliger Quotient
r = a%b;        // Divisionsrest
```

Darum soll ein Rahmenprogramm gelegt werden, das

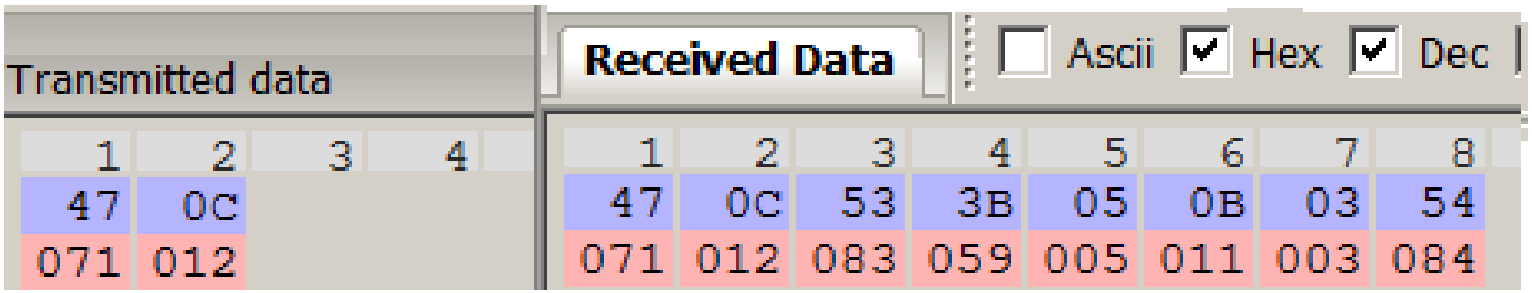
- in einer Endlosschleife
- vom PC auf zwei Bytes für a und b wartet,
- die zu testenden Anweisungen ausführt und
- 8 Bytes (s, d, $2\times$p, q und r) zurückschickt.

3. Modultest vom PC aus

```
#include <avr/io.h>          //test_com_pc.c
#include "com_pc.h"
uint8_t a, b, s, d, q, r; uint16_t p;
int main(){
 com_pc_init();
 while (1){
  a = getByte();  b = getByte();
  //-- zu testende Anweisungen -----
  s = a + b;     //Summe
  d = a - b;     //Differenz
  p = a * b;     //Produkt
  q = a/b;       //ganzzahliger Quotient
  r = a%b;       //Divisionsrest
  //---------------------------------
  sendByte(a);     sendByte(b);
  sendByte(s);     sendByte(d);
  sendByte(q);     sendByte(r);
  sendByte(p>>8); sendByte(p&0xFF);
 }
}
```

Test mit dem HTerm

- Projekt »F4-com_pc« öffnen.
- »Dragon« und Compiler-Optimierung -O0 auswählen.
- Übersetzen. Debugger starten. Programm freilaufend starten.
- HTerm öffnen. 9600 Baud, 8 Datenbit, 1 Stoppbit.
- COM-Port des angesteckten »PmodUSBUART«. »Connect«.
- 2 Byte senden und 8 Bytes empfangen.



a	b	$a+b$	$a-b$	a/b	$a \cdot b$
71	12	83	59	5 Rest 11	$3 \cdot 2^8 + 84 = 852$

Modultest mit Python

Python-Programm für den Test vom PC

```
import serial                    #Programm: test_com_pc.py
ser = serial.Serial("COM9")#COM anpassen!
# Testbeispiele
i_tuple = ((27,87),(220,20),(110,4), (218, 219))
for inp in i_tuple:              #fuer alle Testbeispiele
 ser.write(bytearray(inp))#als Byte-Array versenden
 x = ser.read(8)                 #auf 8 Bytes warten
 a = x[0]; b = x[1]; s = x[2] #Zusammensetzen der
 d = x[3]; q = x[4]; r = (x[5]#empfangen Bytes zu
 p = x[6] * 256 + x[7]           #Datenobjekten
 print('a=%3i , b=%3i Summe: %i %s'%(a,b,s,str(s==a+b)))
 print(13*' '+'Differenz:%i %s'%(d, str(d==a-b)))
 print(13*' '+'Quotient: %i Rest: %i (%s)'%(q, r,
          ... str(a==q*b+r)))
 print(13*' '+'Produkt:  %i (%s)'%(p, str(p==a*b)))
ser.close()
```

(Details der Python-Programmierung siehe später Zusatzteil.)

Testdurchführung

- HTerm »Disconnect«.
- Auf dem Mikrorechner das Programm »test_com_pc« starten.
- Windows-Konsole (cmd.exe) starten. Im Verzeichnis H:\Informatikwerkstatt\Python das Programm test_com_pc.py starten. Programmausgabe kontrollieren.

```
H:\Informatikwerkstatt\Progr_IW\Python>test_com_pc.py
a= 27 b= 87 Summe: 114 True
            Differenz: 196 False
            Quotion: 0 Rest: 27 True
            Produkt: 2349 True
a=220 b= 20 Summe: 240 True
            Differenz: 200 True
            Quotion: 11 Rest: 0 True
            Produkt: 4400 True
a=110 b=  4 Summe: 114 True
            Differenz: 106 True
            Quotion: 27 Rest: 2 True
            Produkt: 440 True
a=218 b=219 Summe: 181 False
            Differenz: 255 False
            Quotion: 0 Rest: 218 True
            Produkt: 47742 True
```

Die Tests zu den Ausgabezeilen mit »False« haben versagt. Wo liegt die Fehlerursache:

- im Testobjekt
- im Testrahmen auf dem PC oder
- im Phyton-Programm?

Typfehler im Testobjekt: »uint8_t d« kann keine negativen Differenzen darstellen.

Zusatzteil Python

Python-Installation für die Übungen[3]

Die Übung nutzt Python3 unter Windows:

- Von »www.python.org« für die neueste »stable« Version (aktuall 3.5.2) »Windows x86-64 executable installer« herunterladen und als Administrator ausführen.
- Haken bei »Add Python 3.5 to Path«.
- Customize installation, Next, Haken bei »Install for all users«.

Nachinstallation der in den Übungen benötigten Zusatzpakete:

- Windows-Taste +"R" > "cmd".
- rechte Maustaste, "Run as Administrator".

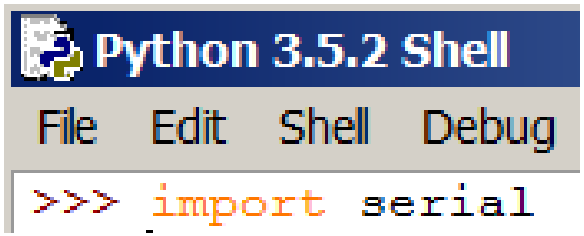
```
pip install pyserial
pip install matplotlib
```

[3] Im Labor sollte bereits alles richtig installiert sein.

Ausprobieren von Programmanweisungen

Python eignet sich u.a. deshalb gut zum Programmierenlernen und für Test-Scripte, weil sich Programmzeilen einzeln auf der Konsole testen lassen. Start der Programmierkonsole »Idle« von

```
> Start > Python 3.5 > IDLE (Python 3.5 ...)
```



Eine der wichtigsten Funktion zum Probieren:

```
print(<Obj>{,Obj})
```

Aufruf mit einer kommaseparierten Liste von Objekten. Ausgabe der Textdarstellungen, getrennt durch Leerzeichen.

Zum Ausprobieren auf der »idle«:

```
>> s = 'Hallo'
>> print(3, 5==7, s) #Zahl, Wahrheitswert, Text
>> 3 False Hallo
>> print(type(s), len(s))
>> <type 'str'> 5
```

Textverarbeitung

Beim Testen sind gut lesbare Textdarstellungen wichtig. :

- unformatierte Textkonvertierung: str(<OBj>)
- formatierte Textkonvertierung: '<Formatstring>'%(<Wertetupel>)
- Verketten von Texten: Text1 + Text 2

Zum Ausprobieren auf der Konsole:

```
>> a=5; b=37.7; s='Hallo'
>> >print(str(a) + str(b) + s) #print(a, b, s)
>> 537.7Hallo
>> print('a=%2i, b=%4.2f: %s'%(a,b,s))
>> a= 5, b=37.70: Hallo
```

("%s" – Text (string); "%ni" – ganze Zahl (int), Darstellung mit mindestens n Zeichen; "%$n.m$f" – Gleitkommazahl (float), Darstellung mit mindestens n Zeichen und m Nachkommastellen).

Interaktive Eingaben

In Testscripten werden Eingaben und Sollausgaben vorzugsweise als Konstanten (Tupel oder Listen siehe nächste Folie) vereinbart oder aus Dateien geladen[4]. Die nachfolgende Eingabeanweisung braucht man vor allem, um Python-Scripte auf dem PC anzuhalten:

```
e = raw_input('<Ausgabetext>')
```

Sie gibt einen Text aus, wartet auf eine Eingabe + <Enter> und hat als Rückgabewert die eingegebene Zeichenkette ohne »Enter«:

```
>> e = raw_input('Warte␣auf␣Eingabe:␣')
>> Warte auf Eingabe: text
>> print('Eingabetext:␣' + e)
>> Eingabetext: text
```

[4]Grund: Tests werden mehrfach wiederholt und das mehrfache Eingeben derselben Werte ist lästig und fehleranfällig.

Sequenzobjekte: Zeichenkette, Tupel

Zeichenkette: Sequenz von Zeichen bzw. Bytes. Auswahl von Elementen und Teilzeichenketten.

```
>> s='Hallo Welt!'
>> print('s[3]="%s", s[4:9]="%s"'%(s[3],s[4:9]))
>> s[3]="l", s[4:9]="o Wel"
```

Tupel: Sequenz beliebiger Objekte.

```
>> t = ('abc', 3, 4.2)
>> print(t, t[1], t[1:2])
>> (('abc', 3, 4.2), 3, (3,))
```

Sequenzobjekte: Listen, Iteratoren

Liste: Sequenz beliebiger Objekte mit zusätzlichen Funktionen zum Einfügen, Löschen und Sortieren von Elementen.

```
>> l = []; print(l) # leere Liste
>> []
>> l.append('E1'); print(l) # 1. Element anfügen
>> ['E1']
>> l.append(3==4); print(l) # 2. Element anfügen
>> ['E1', False]
```

Iteratoren: Funktionen zur Generierung allg. beschreibbarer Folgen.

```
>> print(range(5))
# Erzeugt Folge: 0, 1, 2, 3, 4
>> print(range(-4,2,1))
# Erzeugt Folge: -4, -3, -2, -1, 0, 1
```

Kontrollstrukturen, Fallunterschiedungen

Python unterstützt alle gebräuchlichen Kontrollstrukturen: Fallunterscheidungen, Schleifen, Unterprogramme, ... und eine Spezialstruktur für die Fehlerbehandlung:

```
if <B1>:
 <Anweisungsfolge, wenn B1 erfüllt ist>
elif <B2>:
 <Anweisungsfolge, wenn B1 nicht und B2 erfüllt>
else:
 <Anweisungsfolge, wenn weder B1 noch B2 erfüllt>
<immer auszuführende Anweisungen>
```

- Bedingungen »B1« und »B2« müssen den Typ <type 'bool'> haben. Bildung durch Vergleich, z.B. »a==b«.
- Nach »:« Einrücktiefe erhöhen und bis Blockende beibehalten.
- Hinter der letzten bedingt auszuführenden Anweisung Einrücktief zurücksetzen.

Wiederholschleifen

Wiederholschleifen iterieren über Sequenzobjekte (Zeichenketten, Tupel, Listen):

```
>> s='Hallo'
>> for c in s: # für alle Zeichen in s
... print(c)   # Einrücktiefe erhöhen
...            # Einrücktiefe zurücksetzen
H
a
l
l
o
```

Auch hier nach »:« (für den Schleifenkörper) Einrücktiefe erhöhen und nach der Schleife zurücksetzen.

Zusätzliche Fallunterscheidung im Schleifenkörper:

```
>> i=0;
>> for c in s:
... if c=='l': # Einrücktiefe erhöhen
...  print(i)  # Einrücktiefe erhöhen
    i=i+1      # Einrücktiefe zurücksetzen
...            # Einrücktiefe zurücksetzen
2
3
```

Programme besser als Dateien eingeben und Testen. In der »Idle«:

```
File > New File (Ctrl+N)
```

Programmeingabe:

```
s='Hallo'; i=0;
for c in s:
 if c='l': # Einrücktiefe erhöhen
  print(i) # Einrücktiefe erhöhen
 i=i+1;    # Einrücktiefe zurücksetzen
```

Programm Speichern

`File > Save (Ctrl+S), <Pfad+Dateinamen>`

und starten

`Run > Run Modul (F5).`

Unterprogramme

```
def <Funktionsname>(<Liste  Aufrufparameter>):
 <Anweisungen> # Einrücktiefe erhöhen
 [return <Rückgabewert>]
                # Einrücktiefe zurücksetzen
```

Beispiel:

```
# Dateiname: add.py
def add(a, b):
 return a+b

if __name__=='__main__':  # nur bei Ausführung
 print(add(5, 7))         # nicht bei Import
```

Ausführen der Datei »add.py:

```
>> add.py
12
```

Import, Fehlerbehandlungen

Mit »import« lassen sich Funktionen aus anderen Dateien (Modulen) nutzbar machen. Die Bedingung »__name__'=='__main__'« für die Abarbeitung der Testbeispielausge ist dann »False«:

```
>> import add
>> print(add.add(3,5))
8
```

Wenn sich ein Programm nicht bei einem Ausführungsfehler beenden soll:

```
try:
 <auszuprobierende Anweisungen>
except <abzufangende Fehlertypen>:
 <Anweisungen im Fehlerfall>
```

Ermittlung verfügbarer COM-Ports unter Windows

```
def get_COMs():
 import serial
 port_list = []
 for pnr in range(1, 255):
  port = 'COM%i'%(pnr)
  try:
   s = serial.Serial(port)
   s.close()
   port_list.append(port)
  except (OSError,serial.SerialException):
   pass
 return port_list

if __name__=='__main__': # Testrahmen
 print(get_COMs())
```

Wie könnte man zusätzlich feststellen, an welchem COM-Port der programmierte Mikrorechner angeschlossen ist?

Hausaufgabe

- Handout zum aktuellen Foliensatz noch mal lesen.
- Wiederholungsfragen auf dem nächsten Handouts beantworten.

Aufgaben

Aufgabe 5.1: Modultest Vorzeichenzahlen

Ändern Sie das Mikrorechnerprogramm Folie 31 und das Python-Programm Folie 35 so, dass vorzeichenbehaftete Zahlen addiert, subtrahiert, dividiert und multipliziert werden.

Testbeispiele für das Python-Programm:

```
i_tuple = ((-23,45), (-89,-7), (0x7F, -17),
           (-58, -99))
```

Hinweise:

- Im Web suchen, wie in Python Vorzeichenzahlen in Byte-Strings umgewandelt werden.
- Entwicklung von µP-Funktionen für das Senden und den Empfang von Vorzeichenzahlen.

Aufgabe 5.2: Test einer 2-Byte-Multiplikation

- Schreiben Sie ein Programm, das zwei 2-Byte vorzeichenbehaftete Faktoren empfängt, multipliziert und ein 4-Byte-Produkt zurücksendet.
- Schreiben Sie ein Python-Programm, dass dieses Programm mit zehn zufällig ausgewählten Beispielen testet.

Kontrolle für Testbeispiele mit Sollergebnis ungleich Ist-Ergebnis:

- Fehler im Python-Programm?
- Im Mikrorechner ankommende Daten falsch?
- Rechnet der Mikrorechner falsch?
- Interpretiert Python den Rückgabewert falsch?

Hinweise für die Testdurchführung siehe nächste Folie.

Hinweise zur Fehlersuche

Der Test von Mikrorechnerprogrammen mit serieller Kommunikation im Schrittbetrieb ist dahingehend problematisch, dass

- Mikrorechner- und PC-Programm beim Anhalten Daten verlieren können und
- nach Fortsetzung auf die verlorenen Daten warten.

Vorschlag zur Problemvermeidung

- In beiden Programmen nach jedem versendeten und jedem empfangen Paket einen Unterbrechungspunkt setzen.
- Unterbrechnungspunkte in Python setzt man mit (siehe Zusatzteil Folie 43):

```
raw_input('Enter zur Programmfortsetzung:')
```

- Nach jedem Halt zuerst das empfangende und dann das sendende Programm starten.