

Michael Fothe

# **Problemlösen mit Python**

ThILLM Bad Berka 2002  
Reihe „Materialien“ Heft 72

# Gliederung der vier Skripte

## Python I

1. Vorbemerkungen
2. Variablen
3. Ganze Zahlen und arithmetische Operationen
4. Wertzuweisung
5. Wahrheitswerte und logische Operationen
6. Einfache Anweisungen
7. Funktionen
8. Bruchrechnung mit dem PC
9. Begriff „Algorithmus“
10. Praktikum „Gregorianischer Kalender“
11. Rekursive Funktionen

## Python II

1. Zeichenketten (Strings) und String-Operationen
2. Muster I
3. Muster II
4. Palindrom
5. Prüfbit
6. International Standard Book Number (ISBN)
7. Mengen
8. Dualzahlen
9. Stapel (stack, Keller)
10. Suchen in Texten (Mustererkennung)
11. Datenkompression
12. DNS-Sequenzierung
13. Cäsar-Code
14. Tupel

## Python III

1. Gleitpunktzahlen und arithmetische Operationen
2. Praktikum „Gleitpunktzahlen“
3. Listenverarbeitung
4. Listen in Python
5. Sortierverfahren
6. Praktikum „Listen“
7. Module
8. Praktikum „Modulares Programmieren“
9. Methoden der Software-Entwicklung
10. Modellieren eines Automaten

## Python IV

1. Objektorientiertes Programmieren (OOP)
2. Kapselung
3. Vererbung

- 4. Überladen von Operatoren
- 5. Praktikum „OOP“

# Python I

Gedanken werden dann nur gestaltend und schöpferisch, wenn sie an etwas Vorhandenes anknüpfen.  
(Karl Gutzkow)

# 1. Vorbemerkungen

## a) Wer ist der Erfinder von Python?

Guido van Rossum

## b) Woher kommt der Name Python?

Guido sah sich besonders gern *Monty Python's Flying Circus* im Fernsehen an.

## c) Wird Python von führenden Firmen zur Programmentwicklung genutzt?

Ja. Ivan van Laningham nennt NASA, Yahoo, Red Hat, Infoseek und Industrial Light and Magic.

## d) Reicht es, wenn ein Programm funktioniert?

Nein, es muss auch verständlich sein.

Verwenden Sie daher Kommentare!

Kommentare beginnen mit einem Doppelkreuz (#) und gehen bis zum Zeilenende.

## e) Was ist beim Erarbeiten eines Programms zu beachten?

Erst denken, dann tippen!

Und: Testen Sie jedes Programm umfassend aus, auch wenn Sie dazu nicht besonders aufgefordert werden!

Zusatzaufgaben sind im Text mit Z gekennzeichnet.

## 2. Variablen

Eine Variable wird gekennzeichnet durch:

### **Name, Datentyp, Wert**

Name: note  
Datentyp: Integer (ganze Zahl)  
Wert: zum Beispiel 2

#### **Name:**

Der Name einer Variablen kann Buchstaben, Ziffern und den Unterstrich ( \_ ) enthalten. Ein Name beginnt mit einem Buchstaben oder einem Unterstrich. Python-Schlüsselwörter sind als Namen nicht zulässig. Groß- und Kleinbuchstaben werden unterschieden.

Beispiele:

nummer	zulässig
n12	zulässig
_raum	zulässig
if	nicht zulässig (Python-Schlüsselwort)
10a	nicht zulässig (Zeichenfolge beginnt mit einer Ziffer)

Verwenden Sie aussagekräftige Namen!

#### **Datentyp:**

In Python werden Variablen nicht deklariert. Aus der Verwendung einer Variablen ergibt sich deren Datentyp.

#### **Wert:**

Variablen besitzen keinen bestimmten Anfangswert.

Das Einlesen eines Wertes erfolgt mit der input-Anweisung. Vom Python-System wird ein Eingabefenster geöffnet, in das der Nutzer den Wert tippt. Die Eingabe ist mit ENTER abzuschließen.

Beispiel: `note = input('Note:')`

Die Ausgabe des Wertes einer Variablen erfolgt mit der print-Anweisung.

Beispiel: `print note`

### 3. Ganze Zahlen und arithmetische Operationen

Die ganzen Zahlen (Integer) sind 32 Bit-Zahlen.  
Sie reichen von  $-2147483647$  bis  $2147483647$ .

Große ganze Zahlen enden auf L (zum Beispiel  $27453627364572646L$ ).  
Sie können beliebig groß werden. Nur der Speicher des PC bildet eine Grenze.

Zu den ganzen Zahlen gehören die folgenden arithmetischen Operationen:

#### Einstellige Operationen:

- Vorzeichenumkehrung
- + Identität

#### Zweistellige Operationen:

- + Addition
- Subtraktion
- \* Multiplikation
- / Division ohne Rest
- % Rest der Division      Es gilt:  $(x / y) * y + (x \% y) = x$
- \*\* Potenzieren

+ - \* / % \*\* sind Operatoren.

#### Arithmetische Ausdrücke:

Arithmetische Ausdrücke werden von links nach rechts ausgewertet.  
Die aus der Mathematik bekannten Vorrangregeln der Operatoren werden jedoch berücksichtigt („Punktrechnung geht vor Strichrechnung!“). Runde Klammern können gesetzt werden.

Beispiele:

- $20 * a$       # Die ganze Zahl 20 und die Variable a sind Operanden.
- $x - y - 1$
- $a + b * c$
- $(a + b) * c$
- $256 / 10$
- $256 \% 10$

## 4. Wertzuweisung

*variable = ausdruck*

Abarbeitung:

1. Der Wert des Ausdrucks wird ermittelt.
2. Der ermittelte Wert wird der Variablen zugewiesen.

Beispiele: `n = 1`      # Anfangswertzuweisung  
          `n = n + 2`   # Der Wert von n wird um 2 erhöht.  
          `c = (a-120) * (b+1)`  
          `googol = 10 ** 100L`   # Eine 1 mit 100 Nullen.

**Aufgabe 1:** Was leisten die folgenden drei Anweisungsfolgen?

1. <code>a = b</code> <code>b = a</code>	2. <code>c = a</code> <code>a = b</code> <code>b = c</code>	3. <code>a = a + b</code> <code>b = a - b</code> <code>a = a - b</code>
---	---	---

**Das erste Python-Programm „Addieren zweier Zeiten“**

```
Beispiel:  2 h 40 min 50 s
           + 7 h 35 min 20 s
           -----
           10 h 16 min 10 s
```

```
stunden1 = 2
minuten1 = 40
sekunden1 = 50
```

```
stunden2 = 7
minuten2 = 35
sekunden2 = 20
```

```
summe = sekunden1 + sekunden2
sekunden = summe % 60
uebertrag = summe / 60
```

```
summe = minuten1 + minuten2 + uebertrag
minuten = summe % 60      # In jeder Zeile steht eine Anweisung.
uebertrag = summe / 60
```

Eine Anweisungsfolge:

anweisung 1
anweisung 2
anweisung 3
anweisung 4

Struktogramm

stunden = stunden1 + stunden2 + uebertrag

print stunden, minuten, sekunden

**Aufgabe 2:** Setzen Sie das Programm auf den PC um!

**Aufgabe 3:** Ergänzen Sie in dem Programm input-Anweisungen zum Einlesen der Werte der Variablen stunden1, minuten1 und sekunden1 sowie stunden2, minuten2 und sekunden2!

**Aufgabe 4:** Erweitern Sie das Programm so, dass auch Tage eingegeben und verarbeitet werden können (Verwendung der Variablen tage1 und tage2)!



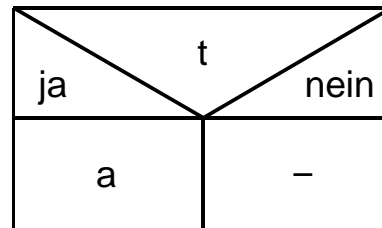
## 6. Einfache Anweisungen

### a) Die bedingte Anweisung (if-Anweisung)

Mit Hilfe von Tests wird entschieden, welche Anweisungsfolge als nächstes abgearbeitet wird. Die Tests sind logische Ausdrücke.

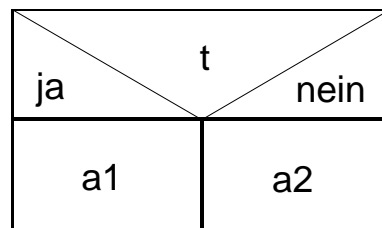
#### 1. Form:

if *test* :  
    *anweisungsfolge*



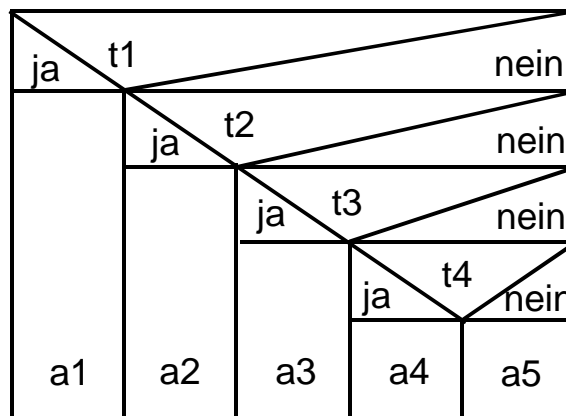
#### 2. Form:

if *test* :  
    *anweisungsfolge1*  
else :  
    *anweisungsfolge2*



#### 3. Form:

if *test1* :  
    *anweisungsfolge1*  
elif *test2* :  
    *anweisungsfolge2*  
elif *test3* :  
    *anweisungsfolge3*  
elif *test4* :  
    *anweisungsfolge4*  
else :  
    *anweisungsfolge5*



Beispiele:

```
a = 0      # FALSCH
b = 1      # WAHR
if a or b :
    print 'ok'
```

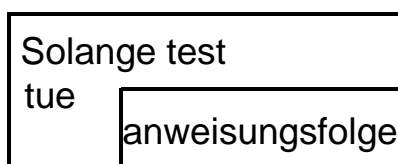
```
x = input('ganze Zahl:')
if x >= 0 :
    print x
else :
    print -x
```

```
anzahl = input('Anzahl:')
if anzahl > 100 :
    print 'groß'
elif anzahl < 10 :
    print 'klein'
else :
    print 'mittel'
```

## b) Die bedingte Schleife (while-Anweisung)

Mit dieser Anweisung können Schleifen mit Abbruchbedingung ausgeführt werden.

```
while test :
    anweisungsfolge
```



Die Anweisungsfolge wird immer wieder abgearbeitet. Die Wiederholung wird beendet, wenn der Test erstmals falsch ist. Ist der Test gleich zu Beginn falsch, wird die Anweisungsfolge nicht abgearbeitet.

Beispiel:

```
n = 1
while n < 30 :
    print n
    n = n + 2
```

### c) Die Laufanweisung (for–Anweisung)

```
for variable in liste :  
    anweisungsfolge
```

Die Variable nimmt nacheinander alle Werte an, die in der Liste stehen.  
Die Anweisungsfolge wird mit diesen Werten abgearbeitet.

Beispiele:

```
for note in [1,2,3,4,5,6] :  
    print note,      # Komma bedeutet: kein Zeilenwechsel.
```

```
for x in [256,128,64,32,16,8,4,2,1] :  
    y = x - 1  
    print y
```

```
L = [+1,0,-1]  
for wert in L :  
    print wert
```

### Verwendung von range

Listen können mit dem Python–Schlüsselwort range automatisch erzeugt werden.

Beispiel:

```
for i in range(10,-8,-2) :           # In der Klammer:  
    print i,                        # Anfangswert, Endwert, Schrittweite
```

Ausgaben:

10 8 6 4 2 0 -2 -4 -6

Hinweis: Der Endwert -8 wird nicht ausgegeben, da er genau „getroffen“ wird.

Für i in Bereich(a,e,s) tue <table border="1"><tr><td>anweisungsfolge</td></tr></table>	anweisungsfolge
anweisungsfolge	

## 7. Funktionen

### a) Definieren einer Funktion

Die Definition einer Funktion besteht aus folgenden Bestandteilen:

1. Python–Schlüsselwort `def`
2. Name der Funktion
3. formale Parameter (in Klammern)
4. Doppelpunkt
5. Anweisungsfolge (eingerückt)

Die Anweisungsfolge enthält mindestens einmal das Python–Schlüsselwort `return`. Nach `return` steht ein Ausdruck. Wird beim Abarbeiten der Anweisungsfolge die `return`–Anweisung erreicht, wird der Wert des Ausdrucks berechnet und die Funktion wird beendet. Der berechnete Wert ist der Funktionswert. Variablen und formale Parameter sind nur innerhalb der Funktion gültig. Sie bilden den lokalen Namensraum der Funktion.

Beispiele:

1. Arithmetische Funktion zum Berechnen des Flächeninhalts eines Rechtecks:

```
def flaeche(a, b) :  
    return a * b
```

2. Logische Funktion zum Überprüfen, ob `t` ein Teiler von `z` ist:

```
def teilbar(z, t) :  
    if z % t == 0 :  
        return 1    # WAHR  
    else :  
        return 0    # FALSCH  
# oder statt der if–Anweisung:  
# return z % t == 0
```

## **b) Aufrufen einer Funktion**

Der Aufruf besteht aus folgenden Bestandteilen:

1. Name der Funktion
2. aktuelle Parameter (in Klammern)

Fortsetzung der Beispiele von a):

zu 1.

```
(x, y) = input('Laenge, Breite:')  
print flaeche(x, y)
```

zu 2.

```
(teiler, zahl) = input('Teiler, Zahl:')  
if teilbar(zahl, teiler) :  
    print 'ja'  
else :  
    print 'nein'
```

## **c) Top-down-Methode**

Die Top-down-Methode ist eine Methode zur Programmentwicklung. Bei deren Anwendung wird durch schrittweise Verfeinerung die passende Struktur von Funktionen gefunden.

Eine Aufgabe wird in Teilaufgaben aufgeteilt. Das Aufteilen kann mehrere Male hintereinander erfolgen. Für jede Teilaufgabe wird eine Funktion definiert.

## **d) Bottom-up-Methode**

Bei der Bottom-up-Methode werden zunächst Teilaufgaben gelöst, mit deren Hilfe dann komplexere Aufgaben bewältigt werden können. Die einzelnen Teillösungen werden von „unten“ nach „oben“ zusammengesetzt, bis die gesamte Aufgabe gelöst ist.

Die Top-down- und die Bottom-up-Methode werden in der Praxis meist kombiniert angewendet.

## 8. Bruchrechnung mit dem PC

Manche Taschenrechner können mit gemeinen Brüchen rechnen. In diesem Abschnitt werden Sie ein Python-Programm zum Bruchrechnen schrittweise erarbeiten. Damit erhalten Sie einen Einblick in die interne Arbeitsweise von Taschenrechnern.

### a) Addieren

Gegeben ist das folgende Python-Programm zum Addieren zweier gemeiner Brüche:

```
z1 = 2           # Zähler des 1. Bruches
n1 = 3           # Nenner des 1. Bruches
z2 = 7           # Zähler des 2. Bruches
n2 = 8           # Nenner des 2. Bruches
zs = z1 * n2 + z2 * n1  # Zähler der Summe
ns = n1 * n2     # Nenner der Summe
print zs, ns
```

**Aufgabe 1:** Setzen Sie das Programm auf den PC um!

### b) Subtrahieren, Multiplizieren und Dividieren

**Aufgabe 2:** Erweitern Sie Ihr Programm um das Subtrahieren, Multiplizieren und Dividieren zweier gemeiner Brüche!

### c) Ermitteln des größten gemeinsamen Teilers (nach EUKLID)

Die folgende while-Anweisung berechnet den ggT von x und y:

```
while y > 0 :
    rest = x % y
    x = y
    y = rest
```

Nach dem Verlassen der while-Anweisung besitzt die Variable x den gesuchten Wert.

In eine Wertbelegungstabelle werden alle Variablen mit den Werten, die sie nacheinander annehmen, aufgenommen.

**Aufgabe 3:** Erarbeiten Sie eine Wertbelegungstabelle für die Berechnung des ggT von 306 und 666!

**Aufgabe 4:** Erarbeiten Sie eine Funktion  $\text{ggT}(x, y)$  zum Berechnen des größten gemeinsamen Teilers von  $x$  und  $y$  und verwenden Sie die Funktion zum Kürzen der Brüche!

#### **d) Ermitteln des kleinsten gemeinsamen Vielfachen**

**Aufgabe Z5:** Erarbeiten Sie eine Funktion  $\text{kgV}(a, b)$  zum Berechnen des kleinsten gemeinsamen Vielfachen von  $a$  und  $b$  und verwenden Sie die Funktion zum Ermitteln des Hauptnenners beim Addieren und Subtrahieren!

Hinweis: Es gilt:  $\text{kgV}(a, b) = a * b / \text{ggT}(a, b)$

## 9. Begriff „Algorithmus“

Ein Algorithmus ist eine Verarbeitungsvorschrift, die aus einer endlichen Folge von eindeutig ausführbaren Anweisungen besteht, mit der man eine Vielzahl gleichartiger Aufgaben lösen kann.

Ein Algorithmus gibt an, wie Eingabegrößen schrittweise in Ausgabe-größen umgewandelt werden.

EVA-Prinzip:   Eingabe → Verarbeitung → Ausgabe

**Aufgabe 1:** Begründen Sie, dass es sich bei der Beschreibung der ggT-Berechnung (nach EUKLID) um einen Algorithmus handelt!

### Das Nim-Spiel

Es sind 31 Streichhölzer gegeben. Die zwei Spieler A und B nehmen abwechselnd mindestens ein Streichholz und höchstens sechs Streichhölzer. Spieler A fängt an. Wer das letzte Streichholz nehmen muss, hat verloren.

Die folgende Strategie garantiert den Sieg von Spieler A:

1. Spieler A nimmt 2 Streichhölzer.
2. Spieler B nimmt 1, 2, 3, 4, 5 oder 6 Streichhölzer.
3. Spieler A ergänzt die von B genommene Anzahl auf 7.
4. Die Schritte 2 und 3 werden wiederholt, bis nur noch ein Streichholz übrig bleibt.
5. Dieses eine Streichholz muss Spieler B nehmen, der dadurch verliert.

**Aufgabe 2:** Stellt die angegebene Strategie einen Algorithmus dar? Begründen Sie Ihre Antwort!

**Aufgabe Z3:** Setzen Sie die angegebene Strategie auf den PC um! Spieler A ist der PC. Spieler B ist der Nutzer des Programms.

## 10. Praktikum „Gregorianischer Kalender“

Bearbeiten Sie die Teile in der angegebenen Reihenfolge!

Zum gregorianischen Kalender:

Im Jahr 1582 wurde von Papst Gregor XIII. eine Kalenderreform angeordnet. Nach den Festlegungen der Reform folgte auf den 4. Oktober 1582 der 15. Oktober. Es wurden also 10 Tage ausgelassen. Außerdem wurde eine neue Regelung für die Festlegung eines Schaltjahres eingeführt. Die Regelung ist auch heute noch gültig. Der gregorianische Kalender wurde in den meisten katholischen Ländern sofort eingeführt, in den protestantischen Ländern Deutschlands erst 1700.

### 1. Teil:

Ein Jahr ist ein Schaltjahr, wenn die Jahreszahl ohne Rest durch 400 teilbar ist oder wenn die Jahreszahl ohne Rest durch 4 und nicht durch 100 teilbar ist.

Beispiele:

Schaltjahre: 1996, 2000, 2004

keine Schaltjahre: 1997, 2003, 2100

Erarbeiten Sie eine logische Funktion `schaltjahr(jahr)`, die ermittelt, ob ein Jahr ein Schaltjahr ist! Der Funktionswert soll 1 sein (WAHR), wenn das Jahr ein Schaltjahr ist. Ansonsten soll der Funktionswert 0 sein (FALSCH).

### 2. Teil:

Die Monate 1, 3, 5, 7, 8, 10 und 12 haben 31 Tage.

Die Monate 4, 6, 9 und 11 haben 30 Tage.

Der Monat 2 hat 29 Tage, wenn er in einem Schaltjahr liegt.

Ansonsten hat er 28 Tage.

Erarbeiten Sie eine Funktion `anzahl(jahr, monat)`, die die Monatslänge ermittelt! Bei der Definition der Funktion ist bei der Bearbeitung des Monats 2 die Funktion `schaltjahr` aufzurufen.

### 3. Teil:

Ein Schaltjahr hat 366 Tage. Ansonsten hat ein Jahr 365 Tage.

Erarbeiten Sie eine Funktion `anzahljahr(jahr)`, die die Jahreslänge ermittelt! Bei der Definition der Funktion ist die Funktion `schaltjahr` aufzurufen.

### 4. Teil:

Die folgenden Daten gibt es nicht: 2003-04-31, 2002-07-42, 2005-02-29 und 2007-15-01.

Erarbeiten Sie eine logische Funktion `existenz(jahr, monat, tag)`, die ermittelt, ob es ein bestimmtes Datum gibt! Bei der Definition der Funktion ist die Funktion `anzahl` aufzurufen.

### 5. Teil:

Der Wochentag für das Datum `jahr-monat-tag` lässt sich mit der Kalenderformel von JACOBSTHAL berechnen:

$$d = \text{tag} + k + j + j / 4 - 2 * (c \% 4)$$

$k$  ist die Monatskennzahl (Schaltjahre in Klammern):

JAN	FEB	MRZ	APR	MAI	JUN	JUL	AUG	SEP	OKT	NOV	DEZ
6(5)	2(1)	2	5	0	3	5	1	4	6	2	4

$j$  ist die Jahreszahl innerhalb eines Jahrhunderts und  $c$  ist die Anzahl der vollen (vergangenen) Jahrhunderte.

Es gilt:  $j = \text{jahr} \% 100$   
 $c = \text{jahr} / 100$

Der Rest der Division des Zwischenergebnisses  $d$  durch 7 ergibt die Nummer des Wochentags (0=Sonntag, 1=Montag, 2=Dienstag usw.).

Berechnen Sie mit der Kalenderformel den Wochentag des heutigen Tages!

Erarbeiten Sie eine Funktion `wochentag(jahr, monat, tag)`, die die Nummer des Wochentags für ein gegebenes Datum ermittelt!

## 6. Teil:

Zum Ermitteln, ob das erste Datum (jahr1, monat1, tag1) zeitlich vor dem zweiten Datum (jahr2, monat2, tag2) liegt, eignet sich die folgende if-Anweisung:

```
if jahr1 < jahr2 :  
    return 1  
elif jahr1 > jahr2 :  
    return 0  
elif monat1 < monat2 :  
    return 1  
elif monat1 > monat2 :  
    return 0  
elif tag1 < tag2 :  
    return 1  
else :  
    return 0
```

Erläutern Sie, wie die if-Anweisung abgearbeitet wird!

Setzen Sie die if-Anweisung als Funktion

```
zeitlich_vor(jahr1, monat1, tag1, jahr2, monat2, tag2)
```

auf den PC um!

## 7. Teil:

Erarbeiten Sie ein Programm, das für ein gegebenes Datum das Datum des nächsten Tages ermittelt!

Bearbeiten Sie dazu die folgenden drei Fälle:

- Der Tag ist nicht der letzte im Monat.
- Der Tag ist der letzte im Monat. Es ist jedoch nicht der 31. Dezember.
- Es ist der 31. Dezember.

## 8. Teil (Z):

Erarbeiten Sie ein Programm, das den Jahreskalender für ein beliebiges Jahr ausgibt!

## 9. Teil (Z):

Erarbeiten Sie ein Programm, von dem das Datum des Ostersonntags für ein beliebiges Jahr ermittelt wird!

Der deutsche Mathematiker, Astronom und Physiker Carl Friedrich Gauß (1777–1855) entwickelte einen Algorithmus zum Berechnen des Osterdatums.

Der Algorithmus lautet in Python:

```
# Die Variable x enthält als Wert eine Jahreszahl.  
k = x / 100  
m = 15 + (3 * k + 3) / 4 - (8 * k + 13) / 25  
s = 2 - (3 * k + 3) / 4  
a = x % 19  
d = (19 * a + m) % 30  
r = d / 29 + (d / 28 - d / 29) * (a / 11)  
og = 21 + d - r  
sz = 7 - (x + x / 4 + s) % 7  
oe = 7 - (og - sz) % 7  
os = og + oe
```

Das Datum des Ostersonntags ergibt sich aus der Variablen os wie folgt:

1. Fall:  $1 \leq os \leq 31$

Das Datum des Ostersonntags ist der os. März.

2. Fall:  $os > 31$

Das Datum des Ostersonntags ist der  $(os - 31)$ . April.

## 11. Rekursive Funktionen

Rekursive Funktionen bestehen aus einer Vereinfachung und einem Abbruch. Eine rekursive Funktion ruft sich selbst auf. Rekursion ist eine Form der Wiederholung.

### a) Fakultät

Diese Funktion spielt in der Mathematik eine wichtige Rolle.

Sie lässt sich rekursiv definieren:

$$0! = 1$$

Wenn  $n > 0$  gilt:  $n! = n * (n-1)!$

Die Definition wird in eine rekursive Funktion übertragen:

```
def faku(n) :  
    if n == 0 :  
        return 1L  
    else :  
        return n * faku(n-1)
```

```
zahl = 5  
ergebnis = faku(zahl)  
print zahl, ergebnis
```

**Aufgabe 1:** Setzen Sie das Programm auf den PC um!

### b) Summe

**Aufgabe 2:** Erarbeiten Sie eine rekursive Funktion sum zum Berechnen der Summe der ersten n positiven ganzen Zahlen!

### c) FIBONACCI-Zahlenfolge

Gegeben ist die folgende rekursive Definition der FIBONACCI-Zahlenfolge:

$$a_0 = 0$$
$$a_1 = 1$$

Wenn  $n > 1$  gilt:  $a_n = a_{n-1} + a_{n-2}$

**Aufgabe 3:** Erarbeiten Sie eine rekursive Funktion fib zum Berechnen von Gliedern der FIBONACCI-Zahlenfolge!

#### d) Ermitteln des größten gemeinsamen Teilers

Die ggT-Berechnung (nach EUKLID) lässt sich auch rekursiv formulieren (siehe Abschnitt 8):

```
def ggT2(x, y) :  
    if y == 0 :  
        return x  
    else :  
        return ggT2(y, x % y)
```

```
a = 12  
b = 40  
g = ggT2(a, b)  
print g
```

**Aufgabe 4:** Setzen Sie das Programm auf den PC um!

#### e) Binomialkoeffizienten

Dem Tafelwerk können Sie wesentliche Informationen zum Berechnen von Binomialkoeffizienten entnehmen.

**Aufgabe Z5:** Erarbeiten Sie ein Programm zum Berechnen von Binomialkoeffizienten!

#### f) Kombinatorik

**Aufgabe Z6:** Erarbeiten Sie ein Programm, in dem die sechs Formeln für Permutationen, Variationen und Kombinationen umgesetzt sind (siehe Tafelwerk)!

#### g) Erzeugen von Dualzahlen

Die Dezimalzahlen werden schrittweise durch 2 dividiert. Die Reste der Division werden in umgekehrter Reihenfolge ausgegeben.

Beispiel: Überführen der Dezimalzahl 28 in die entsprechende Dualzahl

Ab-	↓	28 / 2 = 14	Auf-	↑	28 % 2 = 0
stieg		14 / 2 = 7	stieg		14 % 2 = 0
		7 / 2 = 3			7 % 2 = 1
		3 / 2 = 1			3 % 2 = 1
	↓			↑	

$$1 / 2 = 0$$

$$1 \% 2 = 1$$

Ergebnis: 11100

Das folgende Programm erzeugt die Dualzahl. In dem Programm werden Zeichenketten verwendet (siehe Python II).

```
def dual(zahl) :  
    ganz = zahl / 2  
    rest = zahl % 2  
    if rest == 0 :  
        zeichen = '0'  
    else :  
        zeichen = '1'  
    if ganz == 0 :  
        return zeichen  
    else :  
        return dual(ganz) + zeichen
```

```
n = input('Zahl:')  
print dual(n)
```

**Aufgabe 7:** Setzen Sie das Programm auf den PC um!

### **h) Grundlagen der Arithmetik (nach PEANO)**

Die arithmetischen Operationen lassen sich mit Hilfe von rekursiven Funktionen konstruieren.

1. Funktion (Nachfolger bilden):

$$NA(x) = x+1$$

2. Funktion (Addieren):

$$ADD(x, 0) = x$$

$$\text{Wenn } y > 0 \text{ gilt: } ADD(x, y) = NA(ADD(x, y-1))$$

3. Funktion (Multiplizieren):

$$MUL(x, 0) = 0$$

$$\text{Wenn } y > 0 \text{ gilt: } MUL(x, y) = ADD(x, MUL(x, y-1))$$

4. Funktion (Potenzieren):

$$POT(x, 0) = 1$$

Wenn  $y > 0$  gilt:  $POT(x, y) = MUL(x, POT(x, y-1))$

**Aufgabe Z8:** Setzen Sie die Funktionen auf den PC um!

# Python II

Man glaubt gar nicht, wie schwer es oft ist,  
eine Tat in einen Gedanken umzusetzen!  
(Karl Kraus)

# 1. Zeichenketten (Strings) und String-Operationen

Zeichenketten gehören zu den Sequenztypen. Es handelt sich um höhere Datentypen.

## Zeichenkettenkonstanten:

'Abrakadabra' ist eine Zeichenkettenkonstante.

Doppelte Anführungszeichen können auch verwendet werden.

Zeichenkettenkonstanten können über mehrere Zeilen gehen. Dann sind die Anführungszeichen dreifach zu setzen.

## Einlesen von Zeichenketten:

- mit der `input`-Anweisung: Beim Eintippen der Zeichenkette sind Anführungszeichen zu verwenden.
- mit der `raw_input`-Anweisung: Das Eintippen erfolgt ohne Anführungszeichen.

```
a = input('Zeichenkette:')  
b = raw_input('Zeichenkette:')
```

## Ausgeben von Zeichenketten:

```
print a  
print b
```

## Wertzuweisung:

```
c = 'Sesam, öffne dich!'  
d = 'Impedimenta!'  
e = 'Stupor!'
```

## Aneinanderhängen von Zeichenketten:

```
f = a + b  
g = 100 * c           # Eine Verzweiflungstat!
```

## Vergleich zweier Zeichenketten:

Zeichenketten werden lexikographisch geordnet. Kleiner ist die Zeichenkette, die weiter vorn im Lexikon steht.

Beispiele:

'freundschaft' < 'reich'  
'gross' < 'klein'  
'kleingarten' < 'kleinkind'  
'herein' < 'hereinfallen'

Jedoch gibt es ein Problem: Der Ordnung liegt die Reihenfolge der Zeichen in der ASCII-Tabelle zugrunde (siehe Tafelwerk). Daher kommen erst alle Großbuchstaben, dann die Kleinbuchstaben.

Beispiele:

'A' < 'a'  
'Lexikon' < 'das'

## Ermitteln der Länge einer Zeichenkette:

```
x = 'Zauber'      len(x) == 6  
y = ''           len(y) == 0      # leere Zeichenkette
```

len ist eine vordefinierte Funktion.

## Zugriff auf Zeichen einer Zeichenkette:

Jedes Zeichen einer Zeichenkette z hat einen Index.  
Der Index ist eine ganze Zahl aus dem Intervall von 0 bis len(z)-1.

```
z = 'Enigma'  
z [0] == 'E'  
z [4] == 'm'
```

Mit den folgenden for-Anweisungen wird eine Zeichenkette z zeichenweise ausgegeben:

```
for k in range(0, len(z), 1) :  
    print z [k]
```

```
for zeichen in z :  
    print zeichen
```

## Zugriff auf Abschnitte einer Zeichenkette:

Der Zugriff erfolgt mit der Slice-Operation.

$z[i : k]$  definiert die folgende Zeichenkette:

$z[i] + z[i+1] + z[i+2] + \dots + z[k-1]$

Dabei gilt:  $0 \leq i < k \leq \text{len}(z)$

```
a = 'Donau-Dampfschiffahrtsgesellschaft'  
print a  
b = a[6 : 11]  
print b  
c = a[2 : 3]  
print c  
d = a[6 : len(a)]  
print d  
e = a[6 : ]           # abkürzende Schreibweise für a[6 : len(a)]  
print e
```

## Ändern eines Zeichens einer Zeichenkette:

Wertzuweisungen der Art  $a[4] = 'U'$  sind nicht möglich. Das Ändern eines Zeichens kann mit der Wertzuweisung  $a = a[0 : 4] + 'U' + a[5 : ]$  erreicht werden. (Das kleine u von Donau wird durch ein großes U ersetzt.) Mit solchen Wertzuweisungen können auch Abschnitte einer Zeichenkette geändert werden.

## Funktionen:

Eine Funktion kann eine Zeichenkette als Funktionswert zurückgeben.

## Vordefinierte Funktionen für Zeichen:

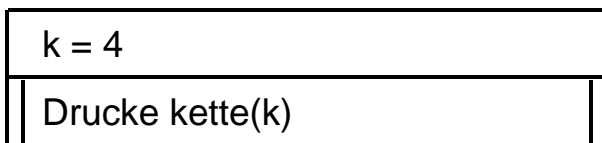
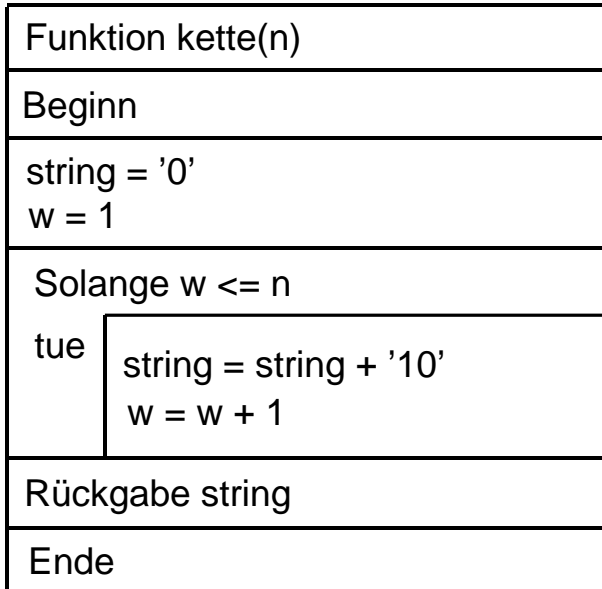
Die Funktion  $\text{ord}(\text{zeichen})$  ordnet einem Zeichen die entsprechende ASCII-Nummer zu. Die Funktion  $\text{chr}(\text{zahl})$  ordnet einer ganzen Zahl das entsprechende ASCII-Zeichen zu.

Beispiele:

```
ord('A') == 65  
chr(65) == 'A'
```

## 2. Muster I

Gegeben ist das folgende Struktogramm:



Das Struktogramm wird in ein Programm überführt:

```
def kette(n) :  
    string = '0'  
    w = 1  
    while w <= n :  
        string = string + '10'  
        w = w + 1  
    return string
```

```
k = 4  
print kette(k)
```

**Aufgabe 1:** Erarbeiten Sie für das Programm eine Wertbelegungstabelle!

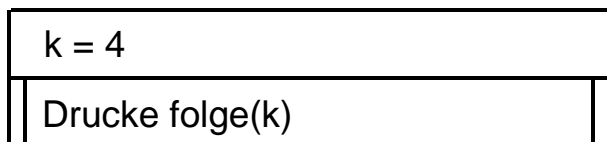
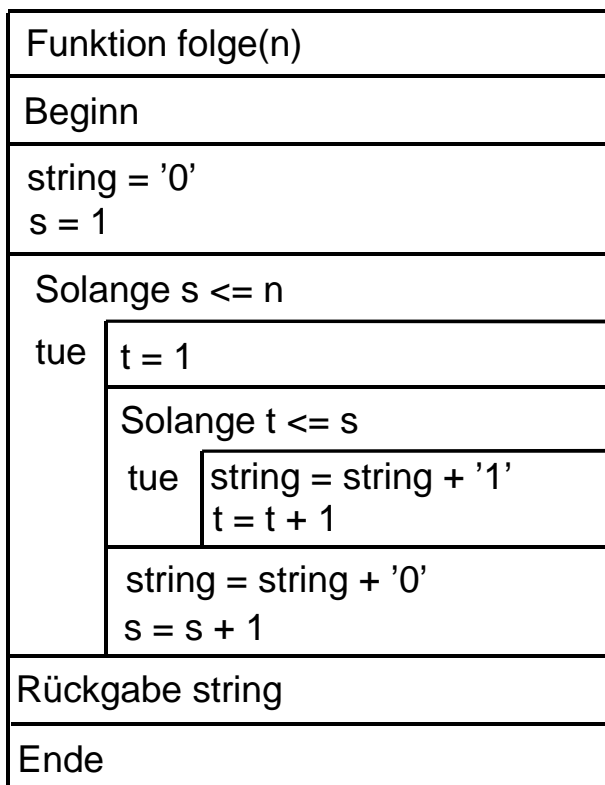
**Aufgabe 2:** Setzen Sie das Programm auf den PC um!

### 3. Muster II

**Aufgabe 3:** Erarbeiten Sie für den folgenden Algorithmus eine Wertbelegungstabelle!

**Aufgabe 4:** Setzen Sie den Algorithmus als Programm auf den PC um!

Struktogramm:



## 4. Palindrom

Eine Zeichenkette, die symmetrisch ist, bezeichnen wir als Palindrom.

Beispiele: 'nebelbeisieleben', 'reliefpfeiler', 'reittier', 'ada', 'otto', 'u', 'bb'

**Aufgabe 5:** Was leistet der folgende Algorithmus?

Funktion test(s)
Beginn
t = "" (leere Zeichenkette)
Für k in Bereich(0, len(s),1)
tue t = s[k] + t
Rückgabe t
Ende

z = 'abcd'
Drucke test(z)

**Aufgabe 6:** Erarbeiten Sie ein Programm, das ermittelt, ob eine Zeichenkette ein Palindrom ist!

## 5. Prüfbit

Ein Bitmuster, das aus sieben Bit besteht, soll in einem Computernetz sicher übertragen werden. Der Sender der Nachricht fügt deshalb dem Bitmuster ein Prüfbit hinzu. Das dabei entstehende Bitmuster soll eine ungerade Anzahl von Einsen enthalten. Der Empfänger der Nachricht überprüft, ob das Bitmuster auch wirklich eine ungerade Anzahl von Einsen enthält. Durch dieses Vorgehen sollen Übertragungsfehler erkannt werden.

Beispiel:

Dem Bitmuster 0110010 ist das Prüfbit 0 hinzuzufügen. Das entstehende Bitmuster 01100100 enthält drei Einsen, also eine ungerade Anzahl.

In den nachfolgenden Aufgaben sind Bitmuster in Zeichenketten zu speichern.

Beispiele:

'0110010', '01100100'

**Aufgabe 7:** Erarbeiten Sie eine Funktion, die die Anzahl an Einsen einer Zeichenkette ermittelt!

**Aufgabe 8:** Erarbeiten Sie ein Programm, das einer Zeichenkette der Länge 7 ein Prüfbit anfügt!

**Aufgabe 9:** Eine Zeichenkette mit Prüfbit wurde in einem Computernetz übertragen. Erarbeiten Sie ein Programm, das ermittelt, ob ein Übertragungsfehler vorliegt! Erkennt Ihr Programm alle möglichen Übertragungsfehler?

## 6. International Standard Book Number (ISBN)

Eine ISBN besteht aus zehn Zeichen (wenn man die Minus-Zeichen nicht mitzählt). Das zehnte Zeichen stellt eine Prüfziffer dar.

Für die ISBN 3897212161 lautet die Prüfziffer 1.

Die Prüfziffer berechnet sich wie folgt:

$$1 * 3 + 2 * 8 + 3 * 9 + 4 * 7 + 5 * 2 + 6 * 1 + 7 * 2 + 8 * 1 + 9 * 6 = 166$$

Prüfziffer (Rest der Division von 166 durch 11): 1

Für den Rest 10 wird X oder x geschrieben.

**Aufgabe 10:** Setzen Sie das folgende Programm in ein Struktogramm um! Das Programm soll Eingabefehler bei der Eingabe einer ISBN erkennen. Dazu berechnet es aus den ersten neun Zeichen die Prüfziffer und vergleicht das Ergebnis mit dem zehnten Zeichen der ISBN.

```
def zahl(zeichen) :
```

```
# Diese Funktion überführt zum Beispiel das Zeichen '3' in die Zahl 3.
```

```
    return ord(zeichen) - ord('0')
```

```
def test(isbn) :
```

```
# 1: korrekt          0: fehlerhaft
```

```
    summe = 0
```

```
    for ziffer in range(1,10,1) :
```

```
        wert = zahl(isbn [ziffer-1])
```

```
        summe = summe + ziffer * wert
```

```
    rest = summe % 11
```

```
    if rest == 10 :
```

```
        if (isbn [9] == 'X') or (isbn [9] == 'x') :
```

```
            return 1
```

```
        else :
```

```
            return 0
```

```
    else :
```

```
        if zahl(isbn [9]) == rest :
```

```
            return 1
```

```
        else :
```

```
            return 0
```

```
nummer = '3897212161'
```

```
print test(nummer)
```

**Aufgabe 11:** Erarbeiten Sie ein Programm, das aus einer ISBN ohne Prüfziffer – das ist eine Zeichenkette der Länge 9 – die Prüfziffer ermittelt und diese an die gegebene Zeichenkette anhängt! Verwenden Sie als Grundlage das Programm von Aufgabe 10.

**Aufgabe Z12:** Erarbeiten Sie ein Programm, das für eine gegebene ISBN feststellt, ob diese syntaktisch korrekt aufgebaut ist!

## 7. Mengen

In diesem Abschnitt werden nur Mengen betrachtet, die aus den Elementen 0, 1, 2, ..., 31 gebildet werden können. Ein Beispiel ist die Menge { 3, 7, 31 }.

Die Mengen werden als Zeichenketten der Länge 32 verwaltet. Die Zeichenketten enthalten nur Nullen und Einsen. Der Menge { 3, 7, 31 } entspricht die Zeichenkette '00010001000000000000000000000001'.

**Aufgabe 13:** Erarbeiten Sie ein Programm, das für zwei Mengen a und b die Durchschnittsmenge d und die Vereinigungsmenge v ermittelt!

Beispiel:

$$a = \{ 0, 12, 13, 14, 30 \}$$

$$b = \{ 4, 13, 30 \}$$

$$d = \{ 13, 30 \}$$

$$v = \{ 0, 4, 12, 13, 14, 30 \}$$

## 8. Dualzahlen

**Aufgabe 14:** Erarbeiten Sie ein Programm, das eine Dualzahl, die als Zeichenkette gegeben ist, in die entsprechende Dezimalzahl überführt!

Beispiel:

Gegeben ist die Dualzahl '110010'. Zum Berechnen der Dezimalzahl wird gerechnet:

$$0 * 2 + 1 = 1$$

$$1 * 2 + 1 = 3$$

$$3 * 2 + 0 = 6$$

$$6 * 2 + 0 = 12$$

$$12 * 2 + 1 = 25$$

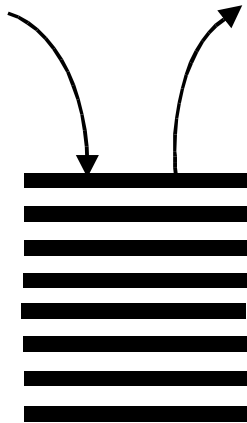
$$25 * 2 + 0 = 50$$

Ergebnis: 50

## 9. Stapel (stack, Keller)

Rekursive Funktionen werden mit Hilfe eines Rekursionsstapels zur Abarbeitung gebracht. Die Werte der lokalen Variablen, die Werte der Parameter und die Rücksprungadresse werden vor dem rekursiven Aufruf der Funktion in dem Stapel gespeichert und nach dem Abarbeiten der Funktion aus dem Stapel geholt. Programmierer und Nutzer eines Programms müssen sich um das Verwalten des Rekursionsstapels nicht selbst kümmern. Diesen Service stellt das Python-System bereit.

Ein Stapel arbeitet nach dem LIFO-Prinzip (Last In – First Out). Das letzte Element, das hinzugefügt wurde, ist stets das erste, das wegzunehmen ist. Von einem Stapel ist nur das oberste Element erreichbar.



Wichtige Stapel-Operationen:

- a) Erzeugen eines leeren Stapels
- b) Stapeln eines Elementes (push)
- c) Lesen des obersten Elementes (top)
- d) Entstapeln eines Elementes (pop)
- e) Test, ob der Stapel leer ist

**Aufgabe 15:** Erarbeiten Sie ein Programm, das die Arbeit eines Stapels simuliert!

## 10. Suchen in Texten (Mustererkennung)

Bei vielen praktischen Anwendungen – zum Beispiel im Internet – ist ein Muster in einem Text zu suchen.

Text:       ersagteabernichts  
Muster:     aber

Bei der Suche wird das Muster so lange von links nach rechts unter dem Text verschoben, bis eine Übereinstimmung vorliegt. Es wird ein Index zurückgegeben (Position des ersten Zeichens bei der ersten Übereinstimmung). Ist das Muster im Text nicht enthalten, wird der Wert  $-1$  zurückgegeben.

Programm:

```
def suche(text, muster) :  
    textlaenge = len(text)  
    musterlaenge = len(muster)  
    for index in range(0, textlaenge - musterlaenge + 1, 1) :  
        if text [index : (index + musterlaenge)] == muster :  
            return index  
    return -1  
  
t = 'ersagteabernichts'  
m = 'aber'  
print suche(t, m)
```

**Aufgabe 16:** Setzen Sie das gegebene Programm auf den PC um!

**Aufgabe 17:** Das Muster kann mehrfach im Text vorkommen. Erarbeiten Sie eine Funktion, die alle Übereinstimmungen von Muster und Text ermittelt! Die Funktion soll als Funktionswert ein Tupel mit allen Positionen liefern (siehe Abschnitt 14).

**Aufgabe Z18:** In der Computer Zeitung Nr. 4 vom 21. Januar 2002 wurde auf Seite 6 berichtet, dass ein Internet-Provider anstößige Ausdrücke wie zum Beispiel „Arsch“ oder „arsch“ aus seinen E-Mail-Adressen verbannt. Verändern Sie das gegebene Programm zum Suchen in Texten so, dass eine 1 oder 0 von der Funktion zurückgegeben wird. Eine 1 bedeutet, dass die E-Mail-Adresse zulässig ist, eine 0, dass sie es nicht ist.

(Das System des Internet-Providers unterscheidet nicht zwischen Namen

und Namensteilen. Pech für alle Arschaks, Barschs, Darscheids,  
Marschalls und Zarschitzkys.)

## 11. Datenkompression

Vor dem Übertragen von Daten im Internet werden diese nach Möglichkeit komprimiert (verdichtet, gepackt). Ein einfacher Kompressionsalgorithmus wird nachfolgend beschrieben.

In der Zeichenkette, die nur aus Nullen und Einsen besteht, werden fünf aufeinanderfolgende Nullen durch ein Plus ersetzt. Fünf aufeinanderfolgende Einsen werden durch ein Minus ersetzt.

Beispiel: Aus der Zeichenkette  $a = '001111111000000000101000000'$  wird die Zeichenkette  $b = '00-11++101+0'$ .

Die Datenkompression erfolgt verlustfrei, d. h. der umgekehrte Vorgang – das Entpacken – liefert die ursprüngliche Zeichenkette.

**Aufgabe 19:** Erarbeiten Sie ein Programm, das eine Zeichenkette nach dem beschriebenen Algorithmus packt! In dem Programm soll die Funktion `suche` aus Abschnitt 10 aufgerufen werden.

**Aufgabe Z20:** Erarbeiten Sie ein Programm, das eine Zeichenkette entpackt!

## 12. DNS-Sequenzierung

Das Modell eines DNS-Moleküls ist eine Zeichenkette, die aus den Buchstaben A, C, G und T zusammengesetzt ist.

Beispiel:

'AACCTGATGGGTACTAAGTCGTAACCGTTGCATGTACGTTA'

DNS-Moleküle können durch Enzyme aufgebrochen werden.

Ein Enzym-Molekül kann immer nur an einer für das Enzym charakteristischen Nukleotidfolge „andocken“ (beispielsweise ATG).

Die DNS wird dann an einer bestimmten Stelle innerhalb dieser Folge getrennt (etwa zwischen A und T). Dieses Beispiel-Enzym könnte durch 'A|TG' beschrieben werden. Die vollständige Zerlegung der DNS-Zeichenkette durch dieses Enzym liefert die Teil-Zeichenketten 'AACCTGA', 'TGGGTACTAAGTCGTAACCGTTGCA' und 'TGTACGTTA'.

**Aufgabe Z21:** Erarbeiten Sie ein Programm, das solche Zerlegungen durchführt!

## 13. Cäsar-Code

Der Klartext besteht aus KlARBuchstaben.

Der Geheimtext besteht aus GeHEIMbuchstaben.

Als Buchstaben sind ausschließlich Kleinbuchstaben zugelassen.

Beim Cäsar-Code gibt es eine feste Zuordnung der Klar- zu den Geheimbuchstaben:

Klarbuchstaben:        abcdefghijklmnopqrstuvwxyz

Geheimbuchstaben:    defghijklmnopqrstuvwxyzabc

Die beiden Alphabete sind um drei Zeichen versetzt.

Beispiel für das Ver- und Entschlüsseln:

Beim Verschlüsseln des Klartextes `synonym` erhält man den Geheimtext `vbqrrqbp`. Beim Entschlüsseln erhält man aus dem Geheimtext wieder den Klartext.

Das Verschlüsselungsverfahren wurde bereits von Gajus Julius Cäsar (100 – 44 v. Chr.) benutzt.

**Aufgabe 22:** Erarbeiten Sie ein Programm, das das Verschlüsseln mit dem Cäsar-Code durchführt!

**Aufgabe 23:** Erarbeiten Sie ein Programm, das das Entschlüsseln mit dem Cäsar-Code durchführt!

**Aufgabe Z24:** Verändern Sie die Programme der Aufgaben 22 und 23 so, dass die beiden Alphabete um eine beliebige Anzahl von Zeichen versetzt sein können! Die Anzahl ist der Schlüssel.

**Aufgabe Z25:** Sender und Empfänger einer Nachricht vereinbaren einen Schlüssel (siehe Aufgabe Z24). Ein Angreifer will den Geheimtext entschlüsseln. Erarbeiten Sie ein Programm, das das unbefugte Entschlüsseln realisiert!

Hinweis: In der deutschen Sprache ist das e der häufigste Buchstabe.

## 14. Tupel

Die Objekte werden durch Wertzuweisungen der Form  $t = (e1, e2, e3)$  definiert. Die Komponenten  $e1$ ,  $e2$  und  $e3$  sind beliebige Datenobjekte. Das leere Tupel ist durch  $()$  und ein Tupel mit genau einer Komponente  $e1$  ist durch  $(e1,)$  definiert (das Komma wegen der Unterscheidung zu einem geklammerten Ausdruck).

Grundsätzlich sind alle Operationen, die bei den Zeichenketten erläutert wurden, auch auf Tupeln ausführbar. Ein Tupel kann von einer Funktion zurückgegeben werden. Komponenten von Tupeln dürfen nicht auf der linken Seite von Wertzuweisungen auftreten.

Beispiel:

```
datum = (2002, 04, 10)
print datum
(jahr, monat, tag) = datum
print jahr
print monat
print tag
datum = datum [0 : 2] + (30,)
print datum
```

# Python III

Was man nicht versteht, besitzt man nicht.  
(Johann Wolfgang von Goethe)

# 1. Gleitpunktzahlen und arithmetische Operationen

In Python gibt es ganze Zahlen (Integer) und Gleitpunktzahlen (Real).

Beispiele:

Python	Art	Mathematik
17	ganze Zahl	17
0	ganze Zahl	0
-23456	ganze Zahl	-23456
2.34	Gleitpunktzahl	2,34
17.0	Gleitpunktzahl	17,0
3.15e-2	Gleitpunktzahl	0,0315
1E3	Gleitpunktzahl	1000,0

In Python wird stets der Dezimalpunkt verwendet. Der Buchstabe e bzw. E wird 'mal 10 hoch' gelesen.

Zu den ganzen Zahlen siehe Python I.

## Zahlenraum von Computern:

Der Zahlenraum ist ein diskreter Raum. Er enthält „Löcher“. Benachbarte Zahlen liegen zum Teil sehr weit voneinander entfernt.

Beispiel:

Die Zahl, die auf 1.00000000001e30 folgt, ist 1.00000000002e30.

Die Differenz der beiden Zahlen beträgt  $10^{19}$ .

## Arithmetische Operatoren:

+   -   \*   /   \*\*

## Vordefinierte Funktionen:

abs(x)	absoluter Betrag von x Beispiel: abs(-3.5) == 3.5
int(x)	x wird in einen Integer-Wert überführt (Weglassen der Ziffern nach dem Dezimalpunkt) Beispiel: int(2.9) == 2
round(x, n)	rundet die Gleitpunktzahl x auf n Stellen nach dem Dezimalpunkt Beispiel: round(2.357, 1) == 2.4
float(x)	x wird in einen Real-Wert überführt

Beispiel: `float(3) == 3.0`

### Wichtige Funktionen im `math`-Modul:

<code>math.sqrt(x)</code>	Quadratwurzel von $x$
<code>math.pow(x, y)</code>	Potenz ( $x$ hoch $y$ )
<code>math.sin(x)</code>	$\sin x$ (Bogenmaß)
<code>math.cos(x)</code>	$\cos x$
<code>math.tan(x)</code>	$\tan x$
<code>math.asin(x)</code>	$\arcsin x$
<code>math.acos(x)</code>	$\arccos x$
<code>math.atan(x)</code>	$\arctan x$
<code>math.exp(x)</code>	$e^x$
<code>math.log(x)</code>	$\ln x$
<code>math.log10(x)</code>	$\lg x$

Am Programmstart ist das Modul `math` zu importieren: `import math`

Beim Aufruf einer Funktion ist ein qualifizierter Name anzugeben.

Beispiel: `math.log(2.5)` ( $\ln 2.5$ )

### Wert und Datentyp von arithmetischen Ausdrücken:

Enthält ein arithmetischer Ausdruck nur Integer-Operanden, so ist der Ausdruck vom Datentyp Integer.

Enthält ein arithmetischer Ausdruck mindestens einen Real-Operanden, so ist der Ausdruck vom Datentyp Real.

Die arithmetischen Ausdrücke `10 / 3` und `10.0 / 3.0` besitzen verschiedene Werte und Datentypen. Der Operator `/` hat unterschiedliche Funktionalität in Abhängigkeit vom Datentyp der Operanden.

Ausdruck	Wert	Datentyp
<code>10 / 3</code>	3	Integer
<code>10.0 / 3.0</code>	3.333333333333	Real

## Test auf Gleichheit:

Der Test

```
x == y
```

ist unsicher, wenn sich x oder y durch Berechnung ergeben.

Die Verwendung des Tests

```
abs(x - y) < 1.0e-10 * abs(y)
```

wird daher empfohlen.

## Erzeugen von Zufallszahlen:

Im Python-System steht der *Whichmann-Hill Pseudo-Zufallszahlen-Generator für Gleitpunktzahlen* zur Verfügung.

Am Programmumfang ist das Modul `whrandom` zu importieren:

```
import whrandom
```

Die Wertzuweisung

```
z = whrandom.random()
```

liefert eine zufällige Gleitpunktzahl z mit  $0.0 \leq z < 1.0$ .

Die Wertzuweisung

```
z = int(n * whrandom.random()) + 1
```

liefert eine ganzzahlige Zufallszahl aus der Menge  $\{ 1, 2, 3, \dots, n \}$ .



## 2. Praktikum „Gleitpunktzahlen“

**Aufgabe 1:** Begründen Sie die Eignung des Tests  $\text{abs}(x - y) < 1.0e-10 * \text{abs}(y)$  zum Feststellen der Gleichheit der Gleitpunktzahlen  $x$  und  $y$ ! Erarbeiten Sie eine Funktion  $\text{gleich}(x, y)$ , die die Gleichheit von  $x$  und  $y$  mit Hilfe dieses Tests überprüft! Im Falle der Gleichheit von  $x$  und  $y$  soll die Funktion den Wert 1 liefern, ansonsten den Wert 0.

**Aufgabe 2:** Erarbeiten Sie eine Funktion  $\text{ungleich}(x, y)$ , die die Ungleichheit von  $x$  und  $y$  überprüft! Bei der Definition der Funktion ist die Funktion  $\text{gleich}$  aufzurufen (siehe Aufgabe 1).

**Aufgabe 3:** Erarbeiten Sie ein Programm, mit dem die Möglichkeiten a) bis e) zum Potenzieren  $z = x^y$  getestet werden! Testen Sie die Möglichkeiten mit den Werten 4, 4.0, 2.4, 0, -4, -4.0 und -2.4 für  $x$  und  $y$ !

a) Verwenden des Operators `**`

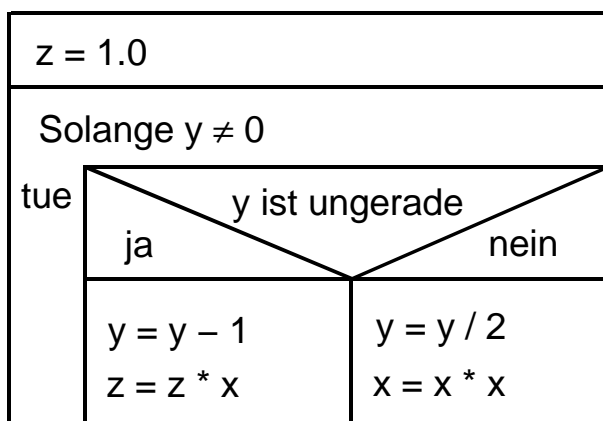
b) Aufruf der Funktion `math.pow(x, y)`

c) Verwenden der Gleichung  $x^y = \text{math.exp}(y * \text{math.log}(x))$

d) Potenzieren durch schrittweises Multiplizieren  
Verwenden einer `while`- oder einer `for`-Anweisung  
Beispiel:  $2.7^5 = 1.0 * 2.7 * 2.7 * 2.7 * 2.7 * 2.7$

e) Potenzieren durch „schnelles Potenzieren“ (nach LEGENDRE 1798)

Das folgende Struktogramm beschreibt den Algorithmus:



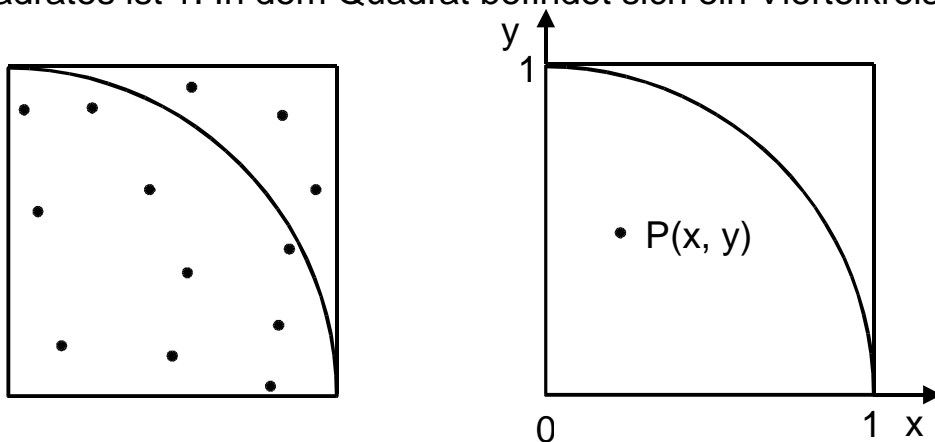
Der Test, ob die ganze Zahl  $y$  ungerade ist, lautet in Python:  
`y % 2 == 1`

**Aufgabe 4:** Erarbeiten Sie ein Programm, das das n-malige Werfen eines Tetraeders simuliert! Mit einem Tetraeder können die Zahlen 1, 2, 3 und 4 geworfen werden. Das Programm soll n einlesen, die Simulation ausführen und dann ausgeben, welche Zahl wie oft geworfen wurde. (Tetraeder zum „Würfeln“ werden mitunter in Spielzeuggeschäften angeboten.)

**Aufgabe 5:** Erarbeiten Sie ein Programm, das die Zahl  $\pi$  näherungsweise mit Hilfe eines Monte-Carlo-Algorithmus ermittelt!

Der Algorithmus wird nachfolgend beschrieben.

Gegeben ist ein Quadrat, auf das es regnet. Die Seitenlänge des Quadrates ist 1. In dem Quadrat befindet sich ein Viertelkreis.



Die Wahrscheinlichkeit dafür, dass ein Regentropfen im Viertelkreis liegt, ist  $\pi / 4$ . Daraus folgt, dass der vierfache Quotient aus der Anzahl an Regentropfen im Viertelkreis und der Anzahl an Regentropfen im Quadrat etwa  $\pi$  ergibt.

Wir legen das Quadrat mit dem Viertelkreis in den 1. Quadranten eines Koordinatensystems. Ein Regentropfen  $P(x, y)$  liegt im Viertelkreis, wenn  $x^2 + y^2 \leq 1$  gilt. Die zufälligen Werte  $x$  und  $y$  werden mit dem Zufallszahlen-Generator erzeugt.

**Aufgabe Z6:** Bei den einfachen Funktionen sind die Parameter Datenobjekte. Bei den Funktionen höherer Ordnung können Parameter auch Funktionsobjekte sein.

Beispiel:

```
def g(x) :                               # einfache Funktion
    return 2.1*x+1.0

def h(x) :                               # einfache Funktion
    return 3.8*x*x-1.5

def uni(f, x) :                          # Funktion höherer Ordnung
    return f(x)

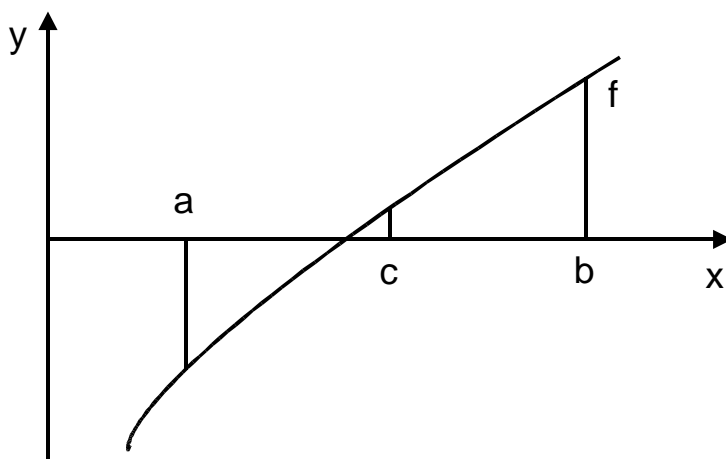
r = 5.2
print uni(g, r)

s = -7.3
print uni(h, s)
```

Setzen Sie das Programm auf den PC um!

**Aufgabe 7:** Bei vielen praktischen Aufgaben sind Nullstellen von stetigen Funktionen zu berechnen:  $f(x) = 0$ . Oft gelingt das jedoch nicht exakt. In solchen Fällen werden Algorithmen zum Ermitteln von Näherungslösungen eingesetzt. Ein solcher Algorithmus ist die Bisektion.

Zuerst werden durch Probieren zwei Startwerte  $a$  und  $b$  ermittelt, für die die Funktionswerte unterschiedliches Vorzeichen besitzen. Im Weiteren gilt  $f(a) < 0$  und  $f(b) > 0$  (siehe Abbildung). Die Nullstelle liegt im Intervall  $(a, b)$ .



Setzen Sie den folgenden Algorithmus zur Bisektion in ein Struktogramm um!

Wiederhole die nachfolgenden Schritte solange,  
wie  $\text{abs}(a - b) \geq \text{epsilon}$  gilt:

Berechne mit der Formel  $c = (a + b) / 2$  die Mitte des  
Intervalls  $(a, b)$ .

Berechne den Funktionswert  $y = f(c)$ .

Gilt  $y > 0$ , dann arbeite die Wertzuweisung  $b = c$  ab.

Andernfalls arbeite die Wertzuweisung  $a = c$  ab.

Gib als Näherungslösung die Mitte des Intervalls  $(a, b)$  aus.

Erarbeiten Sie ein Programm, das eine Nullstelle der Funktion  
 $f(x) = -\ln x + x - 2$  näherungsweise mittels Bisektion berechnet!

Nehmen Sie  $a = 2.5$  und  $b = 4.0$  als Startwerte!

Setzen Sie  $\text{epsilon} = 1.0e-10$ !

**Aufgabe Z8:** Verändern Sie das Programm von Aufgabe 7 nach den  
folgenden Vorgaben:

- Das Berechnen der Nullstelle erfolgt in der Funktion `bisek`.
- Die Funktion `bisek` gibt die Nullstelle als Funktionswert zurück.
- Die Funktion, deren Nullstelle berechnet werden soll, und die beiden Startwerte werden der Funktion `bisek` als Parameter übergeben (siehe Aufgabe Z6).
- In der Funktion `bisek` soll überprüft werden, ob die Bedingungen  $f(a) < 0$  und  $f(b) > 0$  erfüllt sind.

### 3. Listenverarbeitung

Eine Liste ist eine lineare Anordnung von Elementen. Bei den Elementen handelt es sich um Atome oder um Listen. Die einzig wichtige Eigenschaft eines Atoms ist seine Unteilbarkeit. Nachfolgend stehen Großbuchstaben für Atome.

Beispiele für Listen:

[ A , B , C , D , E ]  
[ A ]  
[[ A , [ B , C ] ] , D , [ E , F ] ]  
[[ A ]]

Hinweis: [ A ] und A besitzen unterschiedliche Bedeutung.

Eine rekursive Definition: Eine Liste besteht aus dem Erst-Element und der Rest-Liste oder es ist die leere Liste.

Die leere Liste wird mit NIL bezeichnet. Sie zählt zu den Atomen.

#### Grundlegende Funktionen:

1. Kopf einer Liste:  $Kf(z)$   
hat als Wert das Erst-Element der Liste z.

Beispiele:

z	$Kf(z)$
[ A , B , C ]	A
[ A ]	A
[ A , [ B , C ] ]	
[[ A , B ] , C ]	
[[ A ]]	

2. Rumpf einer Liste:  $Ru(z)$   
 hat als Wert die Rest-Liste der Liste  $z$  (diese ergibt sich durch Streichen des Erst-Elementes der Liste  $z$ ).

Beispiele:

$z$	$Ru(z)$
$[A, B, C]$	$[B, C]$
$[A]$	NIL
$[A, [B, C]]$	
$[[A, B], C]$	
$[[A]]$	

3. Anfügen:  $An(x, y)$   
 hat als Wert eine Liste, die  $x$  als Kopf und  $y$  als Rumpf hat.

Beispiele:

$x$	$y$	$An(x, y)$
$A$	$[B, C]$	$[A, B, C]$
$A$	NIL	$[A]$
$[A, B]$	$[C]$	
$[A]$	$[B]$	
$[A]$	NIL	

Es gilt:  $An(Kf(x), Ru(x)) = x$

4.  $Atom(x)$  ist wahr, wenn  $x$  ein Atom ist.

$Leer(x)$  ist wahr, wenn  $x$  die leere Liste ist.

$Gleich(x, y)$  ist wahr, wenn  $x$  und  $y$  gleiche Atome sind.

Ist  $x$  oder  $y$  kein Atom, dann ist die Funktion nicht definiert.

## Weitere Funktionen:

1. Equal(x, y) ist wahr, wenn x und y gleiche Atome oder gleiche Listen sind.

Angabe eines bedingten Ausdrucks:

$$\text{Equal}(x, y) = (\text{Atom}(x) \rightarrow \text{Atom}(y) \text{ \ae } \text{Gleich}(x, y), \\ \text{Atom}(y) \rightarrow \text{FALSCH}, \\ \text{Equal}(\text{Kf}(x), \text{Kf}(y)) \wedge \text{Equal}(\text{Ru}(x), \text{Ru}(y)))$$

2. Reverse(x) hat als Wert eine Liste, die die Elemente von x in umgekehrter Reihenfolge enthält.

Beispiel:

$$\text{Reverse}([A, B, [C, D]]) = [[C, D], B, A]$$

Angabe eines bedingten Ausdrucks:

$$\text{Reverse}(x) = \text{Rev}(x, \text{NIL}) \\ \text{Rev}(r, s) = (\text{Leer}(r) \rightarrow s, \text{Rev}(\text{Ru}(r), \text{An}(\text{Kf}(r), s)))$$

## 4. Listen in Python

Whenever I think I see a tree,  
I see a string instead. (J. McCarthy)

### Listenkonstanten:

- a) [ 1 , 4 , 9 , 16 , 25 , 36 , 49 ]
- b) [ 1 , [ 4 , 9 ] , 16 ]
- c) [ 0 , [ 1 , [ 2 , [ 3 ] , [ 4 ] ] , [ 5 ] ] , [ 6 , [ 7 , [ 8 ] , [ 9 ] ] , [ 10 ] ] ]
- d) [ ] (leere Liste)
- e) [ 12.34 ]
- f) [ 'xyz' , 3 , 7.0 , ( 13 , 17 , 19 ) ]  
(Bei den Elementen handelt es sich um eine Zeichenkette, eine ganze Zahl, eine Gleitpunktzahl und ein Tupel.)

### Wertzuweisung:

```
a = [ 3 , 4 , 5 , 6 ]  
a = a + [ 7 , 8 ]  
a = a + [ 9 ]
```

### Ermitteln der Länge einer Liste:

```
len(a) == 7
```

### Zugriff auf Listenelemente:

- a) Lesen: w = a [2]
- b) Schreiben: a [2] = 1  
a [2] = 5

### Zugriff auf Teillisten (Slice-Operation):

- a) Lesen: r = a [2 : 5]  
s = a [2 : 3]  
t = a [2 : 2]  
u = a [2 : ]
- b) Schreiben: a [2 : 5] = [ 0 , 1 , 2 ]  
a [2 : 3] = [ ]  
a [4 : ] = [ ]  
a [2 : 2] = [ 11 , 12 , 13 , 14 ]

## 5. Sortierverfahren

### Bubblesort:

Bei diesem Algorithmus zum aufsteigenden Sortieren der Elemente einer Liste werden fortlaufend zwei benachbarte Elemente verglichen und gegebenenfalls vertauscht. In einer logischen Variablen wird festgehalten, ob bei einem Durchlauf durch die Liste wenigstens einmal zwei Elemente vertauscht wurden. Ist dies der Fall, wird die Liste erneut durchlaufen. Andernfalls liegen die Elemente in sortierter Reihenfolge vor.

### Algorithmus:

Zu Beginn setze die Variable vertauscht auf WAHR.

Wiederhole die nachfolgenden Schritte solange, wie beim letzten Durchlauf wenigstens einmal vertauscht wurde:

Setze die Variable vertauscht auf FALSCH.

Vergleiche die 0. mit der 1. Zahl, dann die 1. mit der 2. Zahl, dann die 2. mit der 3. Zahl usw. und zuletzt die vorletzte mit der letzten Zahl und vertausche die beiden verglichenen Zahlen, wenn sie nicht in der richtigen Reihenfolge stehen (das ist der Fall, wenn die erste größer als die zweite ist) und setze die Variable vertauscht auf WAHR.

Der Algorithmus wird in Python umgesetzt:

```
def bubblesort(liste) :
    anzahl = len(liste)      # Anzahl der Elemente der Liste
    vertauscht = 1          # WAHR
    while vertauscht :
        vertauscht = 0      # FALSCH
        for index in range(0, anzahl-1, 1) :
            if liste [index] > liste [index+1] :
                # Dann stehen die beiden Elemente nicht
                # in der richtigen Reihenfolge.
                wert = liste [index]
                liste [index] = liste [index+1]
                liste [index+1] = wert
                # Mit den drei Anweisungen werden die beiden
                # Elemente, die verglichen wurden, vertauscht.
                vertauscht = 1    # WAHR
    return liste

print bubblesort([3,2,12,4,2,9,10,13])
```

## Sortieren durch Mischen:

(Tobias Himstedt, Klaus Mätzel: Mit Python programmieren.  
Einführung und Anwendung skriptorientierter Programmierung.  
dpunkt-Verlag Heidelberg 1999, S. 64 ff.)

Zuerst werden die grundlegenden Funktionen der Listenverarbeitung und die Funktion reverse realisiert:

```
def kf(L) :
    # Kopf der Liste
    if leer(L) :
        print 'Fehler: Liste ist leer.'
    else :
        return L [0]

def ru(L) :
    # Rumpf der Liste
    return L [ 1 : len(L) ]

def an(el, L) :
    # Anfügen am Kopf der Liste
    return [ el ] + L

def leer(L) :
    # Liste leer?
    return L == [ ]

def reverse(L) :
    # Umkehren der Liste
    if leer(L) :
        return [ ]
    else :
        return reverse(ru(L)) + [ kf(L) ]

# Testen der Funktionen:

liste = [ 5 , 4 , 3 , 2 , 1 ]
print liste
print leer(liste)
print kf(liste)
print ru(liste)
liste = reverse(liste)
print liste
```

```
liste = an(0, liste)
```

```
print liste
```

Nun wird das eigentliche Sortieren durch Mischen umgesetzt:

```
def mische(L1, L2) :  
    # Mischen zweier sortierter Listen  
    if leer(L1) :  
        return L2  
    elif leer(L2) :  
        return L1  
    else :  
        if kf(L1) < kf(L2) :  
            return an(kf(L1), mische(ru(L1), L2))  
        else :  
            return an(kf(L2), mische(L1, ru(L2)))
```

```
def zerlege(L) :  
    # Zerlegen einer Liste in zwei Listen  
    if leer(L) :  
        return ( [], [] )  
    elif leer(ru(L)) :  
        return ( [ kf(L) ], [] )  
    else :  
        (m, n) = zerlege(ru(ru(L)))  
        return (an(kf(L), m), an(kf(ru(L)), n))
```

```
def sortiere(L) :  
    # Sortieren einer Liste  
    if leer(L) :  
        return []  
    elif leer(ru(L)) :  
        return [ kf(L) ]  
    else :  
        (M, N) = zerlege(L)  
        return mische(sortiere(M), sortiere(N))
```

```
print sortiere([5,1,7,4,2,6,3])
```

## 6. Praktikum „Listen“

**Aufgabe 1:** Testen Sie die verschiedenen Möglichkeiten des Zugriffs auf Listenelemente und Teillisten!

**Aufgabe 2:** Erarbeiten Sie ein Programm, das das kleinste Element einer Liste ermittelt und ausgibt!

**Aufgabe Z3:** Erweitern Sie das Programm von Aufgabe 2 so, dass auch die Position des kleinsten Elementes ausgegeben wird!

**Aufgabe 4:** Setzen Sie Bubblesort auf den PC um!

**Aufgabe 5:** Setzen Sie die grundlegenden Funktionen der Listenverarbeitung und die Funktion reverse auf den PC um!

**Aufgabe Z6:** Setzen Sie Sortieren durch Mischen auf den PC um! Erläutern Sie die Arbeitsweise der Funktionen mische, zerlege und sortiere!

**Aufgabe Z7:** Erarbeiten Sie eine Funktion rotiere, die eine Liste L um n Elemente rotieren lässt!

Beispiel: Gegeben ist die Liste [11 , 8 , 7 , 4 , 1 , 9 , 3 ].

Die Liste soll um zwei Elemente rotieren.

Resultat ist die Liste [ 7 , 4 , 1 , 9 , 3 , 11 , 8 ].

**Aufgabe Z8:** Erarbeiten Sie eine Funktion spiegele, bei der – in Erweiterung zur Funktion reverse – auch die eingeschachtelten Listen umgedreht werden!

Beispiel: Gegeben ist die Liste [ A , B , [ C , D ] ]. Die Funktion spiegele soll die Liste [ [ D , C ] , B , A ] liefern.

**Aufgabe Z9:** Erarbeiten Sie ein Programm, das die aus anderen Programmiersprachen bekannten Reihungen (Arrays) durch Listen simuliert!

## 7. Module

Beim modularen Programmieren werden Problemlösungen erarbeitet, die aus mehreren Modulen bestehen. Dieses Herangehen unterstützt das Bearbeiten großer Probleme.

Eigenschaften eines Moduls:

- Ein Modul ist logisch oder funktional in sich abgeschlossen.
- Wie ein Modul innen arbeitet, ist außen nicht bekannt (Geheimnisprinzip).
- Ein Modul besitzt Schnittstellen nach außen. Ausschließlich über diese Schnittstellen erfolgt der Zugriff auf das Modul.
- Ein Modul ist überschaubar und somit leicht testbar.
- Ein Modul soll auch bei weiteren Problemlösungen einsetzbar sein.

In Python sind Module Skripte, die eine textuelle Zusammenfassung von Variablen, Funktionen und Klassen darstellen und als Datei mit der Dateierweiterung `.py` im Datei-System abgelegt werden. Die Programmausführung wird durch die Ausführung eines Moduls gestartet.

Beispiel:

```
# Datei list_fkt.py
'Modul enthält grundlegende Funktionen der Listenverarbeitung'

def kf(L) :
    if leer(L) :
        print 'Fehler: Liste ist leer.'
    else :
        return L [0]

def ru(L) :
    return L [ 1 : len(L) ]

def an(el, L) :
    return [ el ] + L

def leer(L) :
    return L == [ ]

def reverse(L) :
    if leer(L) :
        return [ ]
    else :
        return reverse(ru(L)) + [ kf(L) ]

# Modulende
```



Ein Modul importiert die Dienste eines anderen Moduls durch die import-Anweisung. Dafür gibt es zwei Möglichkeiten.

### **1. Möglichkeit: from modulname import \***

Alle im Modul definierten Objekte werden importiert, falls ihr Name nicht mit einem Unterstrich beginnt. Die Namen der importierten Objekte werden in den globalen Namensraum eingetragen, so dass ein Zugriff auf sie möglich wird.

Fortsetzung des Beispiels:

```
# Datei test1.py
from list_fkt import *
liste = [ 5 , 4 , 3 , 2 , 1 ]
print liste
liste = reverse(liste)
print liste
```

Der globale Namensraum wird mit `dir()` angezeigt. Werden mehrere Module importiert und gibt es einen Namen doppelt, so entscheidet die Import-Reihenfolge. Beachten Sie, dass nur das zuletzt importierte Objekt erreichbar ist!

### **2. Möglichkeit: import modulname**

Das Modulobjekt wird als Ganzes importiert. Der Modulname wird in den globalen Namensraum eingetragen. Über qualifizierte Namen kann auf alle im Modul definierten Objekte zugegriffen werden. Es kann auch auf die Objekte zugegriffen werden, die mit einem Unterstrich beginnen. Das sollte jedoch unterbleiben.

Fortsetzung des Beispiels:

```
# Datei test2.py
import list_fkt
liste = [ 5 , 4 , 3 , 2 , 1 ]
print liste
liste = list_fkt.reverse(liste)
print liste
```

Soll ein Modul, das verändert wurde, importiert werden, so ist statt `import modulname` zu verwenden: `reload(modulname)`.

## 8. Praktikum „Modulares Programmieren“

Erarbeiten Sie Module, mit denen die folgenden Aufgaben gelöst werden:

Python I	Bruchrechnung mit dem PC
Python I	Gregorianischer Kalender
Python I	Grundlagen der Arithmetik (nach PEANO)
Python I + II	Überführen von Dezimalzahlen in Dualzahlen und umgekehrt
Python II	Zeichenketten-Operationen
Python II	Prüfbit
Python II	Stapel
Python II	Suchen in Texten
Python III	Bisektion
Python III	Grundlegende Funktionen der Listenverarbeitung und Sortieren durch Mischen

Verwenden Sie bei der Arbeit Quelltexte, die Sie bereits erarbeitet haben.

KISS: Keep it simple and stupid.

## 9. Methoden der Software-Entwicklung

### Einschätzung von Programmen:

Die Qualität eines Programms wird durch die folgenden Kriterien bestimmt:

- **Benutzungsfreundlichkeit:** leichte Handhabung,
- **Zuverlässigkeit:** das Programm arbeitet auch bei ungewöhnlichen Bedienungsmaßnahmen und bei Ausfall gewisser Komponenten des Systems weiter und liefert aussagekräftige Fehlermeldungen,
- **Wartbarkeit:** das Programm kann auch von anderen Programmierern als dem Autor verstanden und korrigiert und auf geänderte Verhältnisse eingestellt werden,
- **Anpassbarkeit:** das Programm kann leicht an weitere Benutzeranforderungen angepasst werden,
- **Portabilität:** das Programm kann ohne größere Änderungen auf unterschiedlichen Computern ausgeführt werden,
- **Effizienz:** ein Problem wird in kurzer Zeit und/oder möglichst geringem Aufwand an Betriebsmitteln gelöst und
- **Ergonomie:** ein interaktives Programmsystem kommt den menschlichen Bedürfnissen der Benutzer nach einer ausgeglichenen Verteilung der geistigen, körperlichen und sozialen Belastungen weitgehend entgegen.

### Phasenmodell:

Die Entwicklung eines Programms ist eine Konstruktion auf ingenieurwissenschaftlicher Grundlage.

In der Literatur werden häufig die folgenden Phasen genannt:

- |                                     |                                  |
|-------------------------------------|----------------------------------|
| • <b>Problemanalyse</b>             | Dokument: Anforderungsdefinition |
| • <b>Entwurf</b>                    | Dokument: Spezifikation          |
| • <b>Implementierung und Testen</b> | Dokument: Quelltext der Module   |
| • <b>Integration und Testen</b>     | Dokument: Programmsystem         |
| • <b>Installation</b>               | Dokument: Produkt                |
| • <b>Wartung</b>                    | Dokument: modifiziertes Produkt  |

Das Ergebnis einer Phase ist ein Dokument.

## Entwicklung von Programmen:

Für schulische Belange lassen sich Entwurf, Implementierung und Reflexion unterscheiden. Die Arbeitsschritte beim strukturierten und modularen Programmieren werden nachfolgend näher erläutert.

Der **Entwurf** besteht im Erstellen einer Lösungsstrategie, die auf ein Abbild des problembezogenen Realitätsausschnittes angewandt wird. Der Realitätsausschnitt wird mit Hilfe von Datenstrukturen modelliert. Die Lösungsstrategie wird in einen Algorithmus umgesetzt, der auf die Datenstrukturen angewandt wird. Zum Entwurf gehört die Analyse der gegebenen Problemstellung.

Die Qualität des späteren Programms sowie die Länge der Implementierungsphase hängen entscheidend von der Genauigkeit der Problemanalyse und Korrektheit des Entwurfs ab. Fehler im Entwurf können in der Implementierungsphase nur noch mit großem Aufwand behoben werden. Die Qualität des Entwurfs bestimmt die Qualität des Produktes.

Der Entwurf ist auch eine Grundlage für die Arbeit im Team (Planung von Schnittstellen, Aufgabenzuweisung, Koordinierung).

Der **Entwurf** lässt sich in drei Phasen gliedern:

### 1. Beschreibungsphase

- Beschreibung der Aufgaben, die zu lösen sind (Wiedergabe der zu lösenden Probleme, Verstehen der Probleme, Angabe der Hilfsmittel)
- Konkretisierungen und begründete Festlegungen (eigene Abgrenzungen der Problematik, eigene Festlegungen mit Begründung, Vorgabe von Möglichkeiten und Grenzen)

### 2. Strukturierungsphase

- Welche Module sind vorgesehen? Was sollen sie leisten?
- Welche grundlegenden Datenstrukturen sind zur Lösung welcher Probleme vorgesehen?
- Welche Funktionen sind vorgesehen? Welche Aufgabe haben sie zu erfüllen? (verbale Beschreibung)

### 3. Algorithmische Phase

- Für jede Funktion sind deren Name und die formalen Parameter anzugeben.
- Es ist zu beschreiben, wie die Funktion die Aufgabe lösen soll.
- Für wesentliche Teile ist der Algorithmus detailliert anzugeben (z. B. als verbale Beschreibung oder Struktogramm).

Die **Implementierung** besteht in der Umsetzung des Entwurfs in ein vom PC ausführbares Programm, im Testen des Programms und im Ergänzen von Kommentaren. Kommentare unterstützen die Lesbarkeit des Quelltextes. Die Kommentare nehmen Bezug auf den Entwurf.

Bei der **Reflexion** wird überprüft, ob das am Anfang formulierte Problem gelöst wurde. Dabei werden die Qualität sowie Einsatzmöglichkeiten und Grenzen des entwickelten Produktes eingeschätzt. Erfahrungen, die bei der Entwicklung des Programms gewonnen wurden, werden aufbereitet.

## 10. Modellieren eines Automaten

(Peter Hubwieser: Didaktik der Informatik. Grundlagen, Konzepte, Beispiele. Springer Berlin Heidelberg New York 2000)

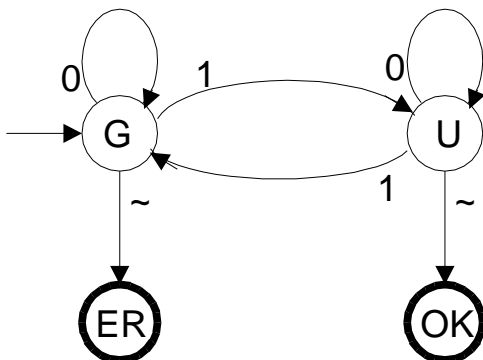
In einem Computernetz wird eine Nachricht übertragen, die aus einer Folge von Bit besteht. Mit Hilfe eines Prüfbit wird erreicht, dass sich in der Nachricht eine ungerade Anzahl an Einsen befindet (siehe Python II, Abschnitt 5). Beim Übertragen der Nachricht können Übertragungsfehler auftreten. Ein Automat soll erkennen, ob ein Fehler vorliegt. Erkennt der Automat einen Fehler, so soll er in den Endzustand ER gehen, andernfalls in den Endzustand OK.

Nachfolgend werden die folgenden vier Ebenen unterschieden:

1. Ebene: Automatenebene
2. Ebene: Algorithmusebene
3. Ebene: Programmebene
4. Ebene: Prozessebene

### Automatenebene

Der Automat wird mit Hilfe eines Graphen beschrieben.



Der Automat besitzt die vier Zustände G (gerade Anzahl an Einsen), U (ungerade Anzahl an Einsen), ER (error, Fehler erkannt) und OK (keinen Fehler erkannt). Der Zustand G ist der Anfangszustand, die Zustände OK und ER sind Endzustände. Die Folge von Bit ist mit dem Zeichen ~ abgeschlossen.

**Aufgabe 1:** Erläutern Sie die Arbeitsweise des Automaten!

**Aufgabe 2:** Geben Sie den Graphen eines Automaten an, der erkennt, ob eine Folge von Bit aus genau acht Bit besteht!

## **Algorithmusebene**

Die Folge von Bit wird in einer Zeichenkette gespeichert. Die Funktion `len` macht das spezielle Abschlusszeichen `~` überflüssig. Zu Beginn wird die Zeichenkette eingelesen und die logische Variable `ok` erhält den Wert 0 (FALSCH). In einer Schleife wird jedes Zeichen der Zeichenkette bearbeitet. Handelt es sich um das Zeichen 1, dann wird der Wert der Variablen `ok` negiert. Andernfalls passiert nichts. Nach dem Bearbeiten des letzten Zeichens der Zeichenkette wird der Wert der Variablen `ok` ausgegeben. Eine 1 (WAHR) steht für „Keinen Übertragungsfehler erkannt.“, eine 0 (FALSCH) für „Fehler!“.

## **Programmebene**

**Aufgabe 3:** Setzen Sie den beschriebenen Algorithmus als Python-Programm auf den PC um (siehe Algorithmusebene)!

## **Prozessebene**

Das Programm wird mit dem Python-Interpreter zur Abarbeitung gebracht. Dann entsteht daraus ein (aktiver) Prozess. Ein Prozess ist eine Folge von Aktionen des PC.

# Python IV

Die Welt und der Himmel bauen sich auf  
aus unseren Denkfehlern.  
(Gerhart Hauptmann)

OOP = ADT + Vererbung + Polymorphie  
(Dahl)

## 1. Objektorientiertes Programmieren (OOP)

1. Im Zentrum der Arbeit stehen Objekte mit bestimmten inneren Zuständen (Werte von Variablen!) und bestimmtem Verhalten (Funktionen!).
2. Zur Kapselung gibt es in Python das Konzept der Klasse. Mit Klassen lassen sich abstrakte Datentypen (ADT) realisieren.
3. Die Definition einer Klasse wird mit dem Python-Schlüsselwort `class` eingeleitet. Jede Klasse erhält einen Namen. (Zusätzlich können Basisklassen angegeben werden – siehe unten!) Anschließend werden Daten- und Funktionsobjekte definiert. Diese werden als Attribute der Klasse bezeichnet.
4. Mit der Definition einer Klasse wird ein neuer lokaler Namensraum definiert (vergleichbar mit dem lokalen Namensraum von Funktionen). Aus der Sicht der Klasse ist der globale Namensraum durch das Modul gegeben, in dem die Klasse definiert ist.
5. Ein Objekt ist eine Instanz (ein Exemplar) einer Klasse. Zur Instanzbildung wird ein parameterloser Funktionsaufruf benutzt. Als Funktionsname dient der Klassenname. Bei der Instanzbildung wird die `__init__` – Methode als Objektkonstruktormethode ausgeführt.
6. Methoden sind Funktionen, die an ein Objekt gebunden sind. Mit Methoden wird der innere Zustand der Objekte geändert.
7. Zur Vererbung wird bei der Definition einer Klasse die Basisklasse angegeben. Die Basisklasse ist der Vorfahr, von dem die neue Klasse abgeleitet wird. Die Daten- und Funktionsobjekte des Vorfahren werden übernommen. Neue Daten- und Funktionsobjekte werden in der Regel hinzugefügt. Gibt es eine Funktion mit dem gleichen Namen wie beim Vorfahren, so wird die alte Funktion überschrieben.

## 2. Kapselung

Im Programm datum1.py erfolgt die Definition der Klasse Jahr.

Die Definition wird mit Zeile # 1 eingeleitet.

In Zeile # 2 beginnt die Angabe der Objektkonstruktormethode `__init__`.

Es handelt sich um eine vordefinierte Spezialmethode (vor und nach `init` jeweils zwei Unterstriche!). Der formale Parameter `self` ist bei dieser Methode anzugeben. Man könnte auch andere Namen nehmen. Der Name `self` hat sich in der Python-Community jedoch eingebürgert (Lesbarkeit des Quelltextes!).

Die `pass`-Anweisung in Zeile # 3 bedeutet, dass bei der Instanzbildung keine weiteren Anweisungen abzuarbeiten sind. Hier könnten zum Beispiel Anfangswertzuweisungen an Variablen erfolgen.

Von außen wird auf Variablen stets mit Hilfe von Methoden zugegriffen.

Ab Zeile # 4 wird eine Methode definiert, die der Variablen `self._jahr` einen Wert zuweist. In Zeile # 5 erfolgt die notwendige Wertzuweisung.

Ab Zeile # 6 wird eine Methode zum Zurückgeben von Werten definiert.

Ab Zeile # 7 bzw. # 8 werden Methoden definiert, die ermitteln, ob das zugewiesene Jahr im Intervall von 1701 bis 3000 liegt und ob es sich um ein Schaltjahr handelt. Beide Methoden liefern das Ergebnis an die rufende Stelle.

```
# datum1.py
```

```
class Jahr: # 1

    def __init__(self): # 2
        pass # 3

    def Einlesen(self, j): # 4
        self._jahr = j # 5

    def Ausgeben(self): # 6
        return (self._jahr, self._korrekt, self._schaltjahr)

    def istKorrekt(self): # 7
        self._korrekt = 1701 <= self._jahr <= 3000
        return self._korrekt

    def istSchaltjahr(self): # 8
        s = (self._jahr % 400 == 0) or \
            ((self._jahr % 4 == 0) and (self._jahr % 100 != 0))
        self._schaltjahr = s
        return self._schaltjahr

# Ende der class-Anweisung

# Skriptende
```

Zum Programm test1.py:

Mit der import-Anweisung in Zeile # 9 werden die vier Methoden der Klasse Jahr importiert.

Mit den Anweisungen in den Zeilen # 10 und # 11 werden die Objekte J1 und J2 erzeugt. Dann werden Meldungen an diese Objekte gesandt.

Die Meldungen werden von den Objekten bearbeitet.

Mit den Funktionsaufrufen ab Zeile # 12 werden den Objekten J1 und J2 die Werte 2001 bzw. 1648 übergeben. Mit den Meldungen ab Zeile # 13 werden die Objekte „aufgefordert“, zu ermitteln, ob es sich um ein zulässiges Jahr und um ein Schaltjahr handelt. Dann werden alle Ergebnisse zurückgegeben.

Die Ausgabe der Zeichen von 'a' bis 'm' dient der Orientierung (siehe unten!).

Ab Zeile # 14 wird das Objekt J3 erzeugt und genutzt.

Ab Zeile # 15 liefern die Objekte J1, J2 und J3 noch einmal alle Ergebnisse.

```
# test1.py

from datum1 import * # 9

J1 = Jahr() # 10
J2 = Jahr() # 11

J1.Einlesen(2001) # 12
J2.Einlesen(1648)

print 'a', J1.istKorrekt() # 13
print 'b', J2.istKorrekt()
print 'c', J1.istSchaltjahr()
print 'd', J2.istSchaltjahr()
print 'e', J1.Ausgeben()
print 'f', J2.Ausgeben()

J3 = Jahr() # 14

J3.Einlesen(2000)

print 'g', J3.istKorrekt()
print 'h', J3.istSchaltjahr()
print 'i', J3.Ausgeben()

print 'k', J1.Ausgeben() # 15
print 'l', J2.Ausgeben()
print 'm', J3.Ausgeben()

# Skriptende
```

Nachfolgend sind die Ausgaben des Programms test1.py angegeben.

```
a 1
b 0
c 0
d 1
e (2001, 1, 0)
f (1648, 0, 1)
g 1
h 1
i (2000, 1, 1)
k (2001, 1, 0)
l (1648, 0, 1)
m (2000, 1, 1)
```

**Aufgabe 1:** Erläutern Sie, wie die Ausgaben zustande kommen!

**Aufgabe 2:** Setzen Sie die Programme datum1.py und test1.py auf den PC um!

**Aufgabe 3:** Dem Objekt J1 soll ein weiterer (anderer) Wert übergeben werden. Ergänzen Sie im Programm test1.py die erforderlichen Anweisungen!

**Aufgabe 4:** Die Methoden istKorrekt und istSchaltjahr sollen nur die Information „erledigt“ (also kein konkretes Ergebnis) liefern. Ändern Sie das Programm datum1.py entsprechend!

**Aufgabe 5:** Die Methode Ausgeben soll auch dann eine Information zum Schaltjahr liefern, wenn das Jahr außerhalb des zulässigen Bereiches liegt. Ändern Sie das Programm datum1.py entsprechend ab!

### 3. Vererbung

Im Programm datum2.py befindet sich ab Zeile # 1 die Klassendefinition Jahr von Abschnitt 2.

Ab Zeile # 2 wird die Klasse Datum definiert. Als Basisklasse (Vorfahr!) ist die Klasse Jahr festgelegt (Angabe erfolgt in Klammern).

Von der Klasse Jahr werden in die Klasse Datum die Variablen self.\_jahr, self.\_korrekt und self.\_schaltjahr sowie die Methode istSchaltjahr übernommen. Zusätzlich gibt es u. a. die Variablen self.\_monat, self.\_tag und self.\_wochentag.

Die Methoden Einlesen (# 3), Ausgeben (# 4) und istKorrekt (# 5) ersetzen die gleichnamigen Methoden aus der Klasse Jahr.

Ab Zeile # 8 wird die Methode Wochentag definiert.

In der Definition der Methode istKorrekt (# 5) werden die Methoden istSchaltjahr und istKorrekt der Klasse Jahr aufgerufen. Die Klasse Jahr ist in diesem Fall beim Aufruf explizit anzugeben (# 6 und # 7).

# datum2.py

```
class Jahr: # 1  
  
    def __init__(self):  
        pass  
  
    def Einlesen(self, j):  
        self._jahr = j  
  
    def Ausgeben(self):  
        return (self._jahr, self._korrekt, self._schaltjahr)  
  
    def istKorrekt(self):  
        self._korrekt = 1701 <= self._jahr <= 3000  
        return self._korrekt  
  
    def istSchaltjahr(self):  
        s = (self._jahr % 400 == 0) or \  
            ((self._jahr % 4 == 0) and (self._jahr % 100 != 0))  
        self._schaltjahr = s  
        return self._schaltjahr  
  
# Ende der class-Anweisung
```

```

class Datum (Jahr): # 2

    def Einlesen(self, j, m, t): # 3
        self._jahr = j
        self._monat = m
        self._tag = t

    def Ausgeben(self): # 4
        return (self._jahr, self._monat, self._tag, self._korrekt, \
                self._wochentag)

    def istKorrekt(self): # 5
        anzahltuplel=(31,28,31,30,31,30,31,31,30,31,30,31)
        t = anzahltuplel [self._monat - 1]
        if Jahr.istSchaltjahr(self) and (self._monat == 2) : # 6
            t = 29
        self._korrekt = Jahr.istKorrekt(self) \
            and (1 <= self._monat <= 12) and (1 <= self._tag <= t) # 7
        return self._korrekt

    def Wochentag(self): # 8
        kennzahltuplel = (6,2,2,5,0,3,5,1,4,6,2,4)
        k = kennzahltuplel [self._monat - 1]
        if Jahr.istSchaltjahr(self) and ((self._monat == 1) \
            or (self._monat == 2)) :
            k = k - 1
        j = self._jahr % 100
        c = self._jahr / 100
        d = self._tag + k + j + j / 4 - 2 * (c % 4)
        d = d % 7
        wochentagtuplel = ('Sonntag','Montag','Dienstag','Mittwoch', \
            'Donnerstag','Freitag','Samstag')
        self._wochentag = wochentagtuplel [d]
        return self._wochentag

# Ende der class-Anweisung

# Skriptende

```

Zum Programm test2.py:

Mit der import-Anweisung in der Zeile # 9 werden die Methoden der Klasse Datum importiert.

Ab Zeile # 10 werden die Objekte D1 und D2 der Klasse Datum und das Objekt J1 der Klasse Jahr erzeugt. In allen drei Fällen wird die Objektkonstruktormethode `__init__` aus der Klasse Jahr abgearbeitet.

Ab Zeile 11 werden Meldungen an die drei Objekte gesandt.

Die Ausgabe der Zeichen von 'a' bis 'm' dient der Orientierung.

```

# test2.py

from datum2 import *                                     # 9

D1 = Datum()                                           # 10
D2 = Datum()
J1 = Jahr()

D1.Einlesen(2001,10,22)                                 # 11
D2.Einlesen(2001,02,29)
J1.Einlesen(1996)

print 'a', D1.istKorrekt()
print 'b', D2.istKorrekt()
print 'c', J1.istKorrekt()

print 'd', D1.istSchaltjahr()
print 'e', D2.istSchaltjahr()
print 'f', J1.istSchaltjahr()

print 'g', D1.Wochentag()
print 'h', D2.Wochentag()
# print 'i', J1.Wochentag()                             # 12

print 'k', D1.Ausgeben()
print 'l', D2.Ausgeben()
print 'm', J1.Ausgeben()

# Skriptende

```

Nachfolgend sind die Ausgaben des Programms test2.py angegeben.

```

a 1
b 0
c 1
d 0
e 0
f 1
g Montag
h Donnerstag
k (2001, 10, 22, 1, 'Montag')
l (2001, 2, 29, 0, 'Donnerstag')
m (1996, 1, 1)

```



**Aufgabe 6:** Erläutern Sie, wie die Ausgaben zustande kommen!

**Aufgabe 7:** Setzen Sie die Programme datum2.py und test2.py auf den PC um!

**Aufgabe 8:** Begründen Sie, dass die Anweisung in Zeile # 12 zu einer Fehlermeldung führt!

**Aufgabe 9:** Definieren Sie eine Klasse Datum1, deren Basisklasse die Klasse Datum ist! Die Klasse Datum1 soll weitere Funktionalität besitzen: Ermitteln des Datums des Vorgänger- und des Nachfolgetages. Die Definitionen der Klassen Jahr und Datum sind bei der Arbeit nicht zu verändern!

**Aufgabe Z10:** Definieren Sie eine Klasse Datum2, deren Basisklasse die Klasse Datum ist! Die Klasse Datum2 soll zu jedem Datum die Nummer ermitteln. Der 1. Januar 1701 soll die Nummer 1 erhalten, der 2. Januar 1701 die Nummer 2 usw. Ein weiteres Beispiel ist der 23. Juni 2002 (90. Geburtstag des britischen Wissenschaftlers Alan Turing). Dieses Datum erhält die Nummer 110112. Die Definitionen der Klassen Jahr und Datum sind bei der Arbeit nicht zu verändern!

## 4. Überladen von Operatoren

Mit Hilfe von Klassen ist es möglich, Operatoren eine besondere Funktionalität zu geben. In dem folgenden Beispiel wird gezeigt, wie der Operator + zum Addieren zweier Zeiten verwendet werden kann (siehe Python I, Abschnitt 4).

```
# zeit.py
```

```
class zeit:
```

```
    def __init__(self, st = 0, mi = 0, se = 0) :  
        self.std = st  
        self.min = mi  
        self.sek = se  
  
    def __add__(self, other) :  
        erg = zeit()  
        summe = self.sek + other.sek  
        erg.sek = summe % 60  
        uebertrag = summe / 60  
        summe = self.min + other.min + uebertrag  
        erg.min = summe % 60  
        uebertrag = summe / 60  
        erg.std = self.std + other.std + uebertrag  
        return erg  
  
    def show(self) :  
        print (self.std, self.min, self.sek)
```

```
# zeitanw.py
```

```
from zeit import *
```

```
a = zeit(2, 40, 50)
```

```
b = zeit(7, 35, 20)
```

```
s = a + b
```

```
print 'Summe:',  
s.show()
```

**Aufgabe Z11:** Setzen Sie das Programm auf den PC um!

## **5. Praktikum „OOP“**

### **1. Team:**

Erläutern Sie Aufbau und Anwendung von vordefinierten Klassen des Python-Systems!

### **2. Team:**

Realisieren Sie einen Stapel und eine Schlange als Liste!

Literatur: Tobias Himstedt, Klaus Mätzel: Mit Python programmieren. Einführung und Anwendung skriptorientierter Programmierung. dpunkt-Verlag Heidelberg 1999, S. 102 ff.

### **3. Team:**

Realisieren Sie das Verarbeiten von römischen Zahlen!

Literatur: Ivan van Laningham: Jetzt lerne ich Python. Markt + Technik Verlag München 2000, S. 233 ff.

### **4. Team:**

Realisieren Sie das Verwalten von Gemeinden und Städten!

Literatur: Martin von Löwis, Nils Fischbeck: Das Python-Buch. Addison-Wesley Bonn 1997, S. 17 ff.

### **5. Team:**

Realisieren Sie ein OOP-Programm nach einer anderen Vorlage! Das Programm soll Kapselung und Vererbung verdeutlichen.