Tutorial: Scipy und Sympy

Version 0.0.1

Aktualisiert am 07.12.2014

Bernhard Grotz

http://www.grund-wissen.de

Die Quelldateien dieses Buchs wurden unter Linux mittels Vim und Sphinx, die enthaltenen Graphiken mittels Inkscape erstellt. Der Quellcode sowie die Graphiken können als Zip-Dateien von der Projektseite heruntergeladen werden:

http://www.grund-wissen.de

Bei Fragen, Anmerkungen und Verbesserungsvorschlägen bittet der Autor um eine kurze Email an folgende Adresse:

info@grund-wissen.de

Augsburg, den 7. Dezember 2014.

Bernhard Grotz

Diese Sammlung an Übungsaufgaben und Lösungen (mit Quellcode) stellt eine in Python 3 geschriebene Version einer Vielzahl an ursprünglich für Maxima entwickelten Beispielaufgaben dar.

Bei dieser Aufgabe handelt es sich um eine Knobelaufgabe bzw. um eine einfache Übungsaufgabe für lineare Gleichungssysteme.

Aufgabe:

Die Aufgabe besteht darin, eine dreistellige Zahl zu finden, die folgende Bedingungen erfüllt:

1. Die Ziffernsumme einer dreistelligen Zahl ist gleich 18.
2. Die Hunderterstelle ist um 6 größer als das 2-fache der Zehnerstelle.
3. Die Einerstelle ist um 6 größer als das 3-fache der Zehnerstelle.

Lösung:

Definiert man die Variablen h als die Hunderterziffer, z als die Zehnerziffer und e als die Einerziffer, so ist die Ziffernsumme gleich $z+h+e$. Aus der Aufgabenstellung lassen sich damit folgende drei Gleichungen aufstellen:

1. Die Ziffernsumme einer dreistelligen Zahl ist 18:

$$z + h + e = 18$$

2. Die Hunderterstelle ist um 6 größer als das 2-fache der Zehnerstelle.

$$h - 6 = 2 \cdot z$$

3. Die Einerstelle ist um 6 größer als das 3-fache der Zehnerstelle.

$$e - 6 = 3 \cdot z$$

Dieses Gleichungsystem kann mittels sympy gelöst werden. Der Code dazu lautet folgendermaßen:

```
import sympy as sy

# Sympy-Variablen initiieren:
h, z, e = sy.S( 'h z e'.split() )

# Gleichungssystem formulieren:
equations = [
    sy.Eq( z + h + e , 18  ),
    sy.Eq( h - 6     , 2*z ),
    sy.Eq( e - 6     , 3*z ),
]

# Gleichungssystem lösen:
sy.solve(equations)

# Ergebnis: {h: 8, z: 1, e: 9}
```

Die Hunderterziffer ist gleich 8, die Zehnerziffer gleich 1 und die Einerziffer gleich 9. Die gesuchte Zahl lautet somit 819.

Bei dieser Aufgabe handelt es sich um eine Knobelaufgabe bzw. um eine einfache Übungsaufgabe für lineare Gleichungssysteme.

Aufgabe:

Die Problemstellung lautet eine Zahl x zu finden, die, wenn sie durch $(x-a)$ geteilt wird, eine gegebene Zahl b ergibt (a sei ebenfalls ein bekannter, konstanter Wert). Die Aufgabe soll für $a = 10$ und $b = 1\frac{5}{21}$ gelöst werden.

Lösung:

Es muss prinzipiell folgende Gleichung gelöst werden:

$$\frac{x}{x-a} = b$$

Für $a = 10$ und $b = 1\frac{5}{21}$ lautet die Gleichung konkret:

$$\frac{x}{x-10} = 1\frac{5}{21}$$

Diese Gleichung kann bereits ohne weitere Vereinfachungen mittels sympy gelöst werden. Der Code dazu lautet folgendermaßen:

```
import sympy as sy

# Sympy-Variablen initiieren:
x    = sy.S( 'x' )
a, b = sy.S( [10, 1+5/21] )

# Gleichung formulieren:
equation = sy.Eq( x/(x-a) , b )

# Gleichung lösen:
sy.solve(equation)

# Ergebnis: [52.0000000000000]
```

Das Ergebnis der Aufgabe lautet somit $x = 52$.

Bei dieser Aufgabe geht es um eine Volumensberechnung beziehungsweise um die Berechnung eines bestimmten Integrals

Aufgabe:

Das Volumen eines Rotationskörpers soll berechnet werden. Gegeben sind die erzeugende Funktion, die Untergrenze und die Obergrenze des Intervalls. Die Rotation soll um die x-Achse erfolgen. Die Aufgabe soll für die Funktion $f(x) = x^2+x+\frac{1}{x}$ gelöst werden, wobei die Untergrenze bei $a = 1$ und die Obergrenze bei $b = 5$ liegen soll.

Lösung:

Der Graph der Funktion $f(x) = x^2 + x + \frac{1}{x}$ kann mit Hilfe von numpy und der matplotlib geplottet werden:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# Wertereihen erzeugen:
x = np.arange(start=1,stop=5, step=0.01)
y = x**2 + x + 1/x

# Funktion plotten:
plt.plot(x,y)

# Layout anpassen:
plt.axis([0,6,0,35])
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.grid(True)

plt.text(3.5, 30, "$f(x)=x^2 + x + 1/x$")

plt.show()
```

Das Ergebnis sieht folgendermaßen aus:

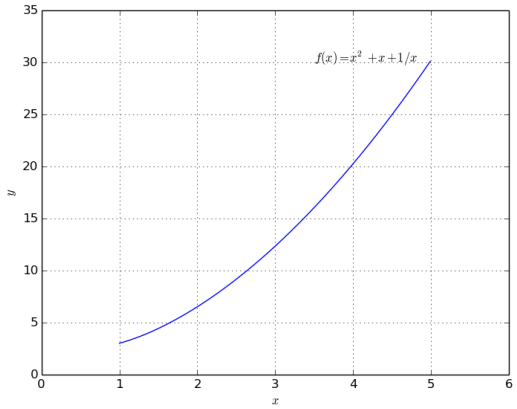

Abbildung 1: Graph der Funktion $f(x) = x^2 + x + \frac{1}{x}$.

Um das Volumen des aus $f(x)$ bei Rotation um die x-Achse entstehenden Rotionskörpers zu berechnen, kann man sich diesen aus lauter winzig schmalen (Zylinder-)Scheiben zusammengesetzt denken. Jede dieser Scheiben hat als Radius den Wert $f(x)$, als Höhe den Wert $\mathrm{d}x$ und als Volumen somit $\pi \cdot r^2 \cdot h = \pi \cdot f(x)^2 \cdot \mathrm{d}x$. Alle diese infinitesimalen Volumina müssen aufsummiert werden, was folgendem Integral entspricht:

$$V = \int_a^b \left(\pi \cdot f^2(x)\right) \cdot \mathrm{d}x = \int_1^5 \pi \cdot \left(x^2 + x + \frac{1}{x}\right)^2 \cdot \mathrm{d}x$$

Dieses Integral kann mittels sympy berechnet werden. Der Code dazu lautet folgendermaßen:

```
import sympy as sy

# Sympy-Variablen initiieren:
x    = sy.S( 'x' )
a, b = sy.S( [1, 5] )

# Bestimmtes Integral berechnen:
sy.integrate( sy.pi * (x**2 + x + 1/x) , (x,a,b) )

# Ergebnis: 15164*pi/15

# Alternativ: Ergebnis als Fließkommazahl ausgeben:
sy.integrate( sy.pi * (x**2 + x + 1/x) , (x,a,b) ).evalf()

# Ergebnis: 3175.94073326904
```

Das Volumen des Rotationskörpers beträgt somit rund $3174,94$ Volumeneinheiten.

Bei dieser Aufgabe geht es darum, eine physikalische Bewegungsgleichung zu lösen.

Aufgabe:

Zwei Flugzeuge verlassen einen Flughafen zur selben Zeit in entgegengesetzte Richtungen mit den Geschwindigkeiten $v_1 = 490\,\mathrm{km/h}$ beziehungsweise $v_2 = 368\,\mathrm{km/h}$. Nach welcher Zeit t haben sie einen Abstand von $s = 3805\,\mathrm{km}$ erreicht?

Lösung:

Die beiden Flugzeuge bewegen sich mit der Summe ihrer Geschwindigkeiten, also mit $v = v_1 + v_2 = 858\,\mathrm{km/h}$ auseinander. Aus der Weg-Zeig-Formel $s = v{\cdot}t$ für Bewegungen mit konstanter Geschwindigkeit lässt sich die gesuchte Größe t berechnen. Der sympy-Code dazu lautet:

```
import sympy as sy

# Sympy-Variablen initiieren:
s = sy.S( 3805 )
v = sy.S(  858 )
t = sy.S(  't' )

# Gleichung formulieren:
equation = sy.Eq( s , v * t )

# Gleichung lösen:
result = sy.solve(equation)

# Ergebnis: [3805/858]

# Ergebnis als Fließkommazahl ausgeben:
float(result[0])

# Ergebnis: 4.434731934731935
```

Die gesuchte Zeit beträgt somit rund $4,435$ Stunden. Etwas eleganter ist allerdings die Angabe in Stunden und Minuten. Sie kann aus dem obigen Ergebnis folgendermaßen berechnet werden:

```
import math

result_f = float(result[0])

hours = math.floor(result_f)
# Ergebnis: 4.0

minutes = (result_f - math.floor(result_f)) * 60
# Ergebnis: 26.083916083916083
```

Die gesuchte Zeit beträgt somit rund 4 Stunden und 26 Minuten.

Bei dieser Aufgabe handelt es sich um ein einfaches Dreisatzproblem.

Aufgabe:

Eine Anzahl von $n_1 = 8$ Bauarbeitern, alle mit gleicher Leistung, benötigt $t_1 = 87$ Tage, um ein Haus zu bauen. Wie viele Tage t_2 sind zum Hausbau nötig, wenn $n_2 = 24$ Bauarbeiter mit der selben Leistung daran arbeiten?

Lösung:

Diese Dreisatzaufgabe lässt sich als einfach Verhältnisgleichung darstellen. Da die insgesamt benötigte Zeit als indirekt proportional zur Anzahl der Arbeiter angenommen wird, ist das Verhältnis $\frac{t_1}{t_2}$ der benötigen Zeiten gleich dem Verhältnis $\frac{n_2}{n_1}$ der Arbeiterzahlen:

$$\frac{t_1}{t_2} = \frac{n_1}{n_2}$$

Diese Gleichung lässt sich auch ohne Computer-Algebra-System leicht nach t_2 auflösen (insbesondere, wenn man auf beiden Seiten die Kehrwerte bildet, d.h. die Gleichung $\frac{t_2}{t_1} = \frac{n_1}{n_2}$ betrachtet). Dennoch soll an dieser Stelle die Aufgabe als Beispiel für vielfach vorkommende Dreisatzaufgaben mit sympy gelöst werden:

```
import sympy as sy

# Sympy-Variablen initiieren:
n1, n2 = sy.S( [8, 24] )
t1 = sy.S( 87 )
t2 = sy.S( 't2' )

# Gleichung formulieren:
equation = sy.Eq( t1/t2 , n2/n1 )

# Gleichung lösen:
result = sy.solve(equation)

# Ergebnis: [29]
```

Die gesuchte Zeit beträgt somit $t_2 = 29$ Tage.

Bei dieser Aufgabe geht es um das Lösen einer Summengleichung.

Aufgabe:

Betrachtet wird eine Folge x_n von aufeinander folgenden, ganzzahligen Werten ($x, n \in \mathbb{N}$).

Die Summe $\sum_{i=1}^{m_1} x_\mathrm{i}$ der ersten m_1 Zahlen sei um einen Differenzwert d kleiner als die Summe $\sum_{i=m_1+1}^{n} x_\mathrm{i}$ der restlichen Zahlen. Wie lässt sich diese Aufgabe mathematisch formulieren? Welche Lösung ergibt sich für $n = 5$, $m_1 = 2$ und $d = 42$?

Lösung:

Damit die obige Bedingung erfüllt ist, muss folgende Gleichung gelten:

$$\sum_{i=1}^{m_1}(x_i) + d = \sum_{i=m_1+1}^{n}(x_i)$$

Um diese Aufgabe mit `sympy` zu lösen, können beide Summen in der obigen Gleichung auch in folgender Form dargestellt werden:

$$\sum_{i=1}^{m_1}(x + i) + d = \sum_{i=m_1+1}^{n}(x + i)$$

Hierbei ist der zu bestimmende Initialwert x um 1 kleiner als der erste Wert der Zahlenreihe. Diese Darstellung hat den Vorteil, dass die Summen leichter formuliert werden können und die Gleichung nur noch eine Unbekannte aufweist. Der Quellcode zur Lösung der Gleichung mittels sympy kann beispielsweise so aussehen:

```
import sympy as sy

# Sympy-Variablen initiieren:
n,m1,d = sy.symbols('n m1 d')
x,i = sy.symbols('x i')

# Terme festlegen
s1 = sy.summation(x + i, (i,1,m1))
s2 = sy.summation(x + i, (i,m1+1,n))

# Gleichungen formulieren:
equation = sy.Eq( s1 + d , s2)
equation_concrete = equation.subs({n:5,m1:2,d:42})

# Gleichung(en) lösen:
sy.solve(equation, x, 'dict')
sy.solve(equation_concrete, x, 'dict')

# Ergebnisse:

# Allgemein:
# [{x: (-d - m1**2 - m1 + n**2/2 + n/2)/(2*m1 - n)}]

# Konkret:
# [{x:33}]
```

Beim Lösen der Gleichung wurde hierbei explizit die Variable x als gesuchte Variable angegeben; ebenso könnte die Gleichung beispielsweise nach d für einen gegebenen Startwert x aufgelöst werden.

Das Ergebnis für die konkrete Aufgabe lautet $x = 33$. In diesem Fall sind die $n = 5$ Zahlen also gleich $(34, 35, 36, 37, 38)$, die Summe der ersten $m_1 = 2$ Zahlen ist $s_1 = 34 + 35 = 69$, die Summe der weiteren Zahlen ist gleich $36 + 37 + 38 = 111$.

Bei dieser Aufgabe geht es um die Formulierung und das Lösung einer Funktionsgleichung.

Aufgabe:

Wenn K. heute $m = 3$ mal so alt wäre wie vor $n = 6$ Jahren, dann wäre K. nun $j = 38$ Jahre älter. Wie alt ist K. heute?

Lösung:

Die gesuchte Variable x gebe das heutige Alter von K. an. Dann lässt sich aus den obigen Bedingungen folgende Gleichung aufstellen:

$$m * (x - n) = x + j$$

Diese Gleichung kann folgendermaßen mit sympy gelöst werden:

```
import sympy as sy

# Sympy-Variablen initiieren:
x = sy.S( 'x' )
m,n,j = sy.S([3, 6, 38] )

# Gleichung formulieren:
equation = sy.Eq( m * (x-n) , x + j )

# Gleichung lösen:
sy.solve(equation)

# Ergebnis: [28]
```

K. ist somit heute 28 Jahre alt.

Bei dieser Aufgabe geht es darum, die Diskriminante einer quadratischen Gleichung zu bestimmen.

Aufgabe:

Gegeben sei die quadratische Gleichung $f(x) = x^2 - 8 \cdot x - 15$. Wie lautet die Diskriminante dieser Gleichung?

Lösung:

Die Diskriminante D einer quadratischen Gleichung $f(x) = a \cdot x^2 + b \cdot x + c$ lässt sich in Abhängigkeit der Parameter a, b und c folgendermaßen bestimmen:

$$D = b^2 - 4 \cdot a \cdot c$$

Die Aufgabe kann folgendermaßen mit sympy gelöst werden:

```
import sympy as sy

# Sympy-Variable initiieren:
x = sy.S( 'x' )

# Gleichung formulieren:
f = x**2 - 8*x - 15

# Koeffizienten a, b und c bestimmen:

a = f.coeff(x, n=2)
b = f.coeff(x, n=1)
c = f.coeff(x, n=0)

D = b**2 - 4*a*c

# Ergebnis: 124
```

Die Diskriminante ist positiv, somit hat die Gleichung die zwei Lösungen $x_{1,2} = \frac{-b \pm \sqrt{D}}{2 \cdot a}$:

```
x1 = ( -b + sy.sqrt(D) ) / (2 * a)
x2 = ( -b - sy.sqrt(D) ) / (2 * a)

x1.evalf()
# Ergebnis:  9.56776436283002

x2.evalf()
# Ergebnis: -1.56776436283002
```

Der Funktionsgraph als Ganzes kann mittels numpy und matplotlib folgendermaßen erzeugt werden:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# Wertereihen erzeugen:
x = np.arange(-3,11,0.01)
y = x**2 - 8*x - 15

# Funktion plotten:
plt.plot(x,y)
```

```
# Layout anpassen:
plt.axis([-3,11,-35,35])
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.grid(True)

plt.text(7, 15, "$f(x)=x^2 - 8x - 15$")

plt.show()
```

Das Ergebnis sieht so aus:

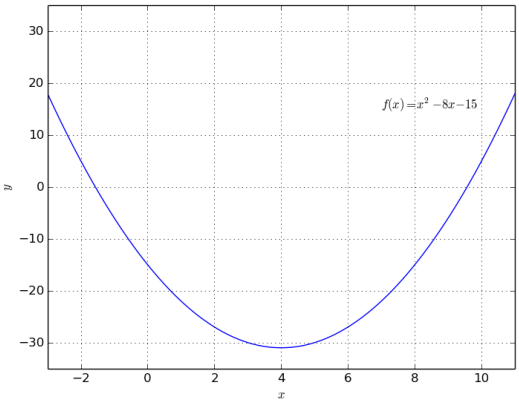

Abbildung 2: Graph der Funktion $f(x) = x^2 - 8 \cdot x - 15$.

Bei dieser Aufgabe geht es um die Formulierung und das Lösung einer geometrischen Funktionsgleichung.

Aufgabe:

Gegeben ist die Diagonale $d = 3 \cdot \sqrt{2}$ eines Quadrats. Wie lässt sich daraus die Fläche A des Quadrats berechnen?

Lösung:

Allgemein gilt für die Diagonale d eines Quadrats in Abhängigkeit von der Seitenlänge a:

$$d = \sqrt{2} \cdot a$$

Umgekehrt gilt somit für die Seitenlänge a in Abhängigkeit von d:

$$a = \frac{d}{\sqrt{2}}$$

Somit lässt sich aus d die Seitenlänge a und damit die Fläche $A = a^2$ des Quadrats berechnen. Ein Beispielcode mit sympy kann beispielsweise so aussehen:

```
import sympy as sy

# Sympy-Variablen initiieren:
d = sy.S( 3 * sy.sqrt(2) )
a = sy.S( 'a' )

# Seitenlänge berechnen:
a = d / sy.sqrt(2)

# Fläche berechnen:
A = a**2

# Ergebnis: 9
```

Die Fläche des Quadrats beträgt somit 9 Flächeneinheiten.

Diese Aufgabe war im Original dafür vorgesehen, um in Maxima die Iteration über Listenelemente zu demonstrieren. Bei der Verwendung von `python` und `sympy` als Computer-Algebra-System hingegen genügt dafür bereits die Python-Standardbibliothek.

Aufgabe:

Gegeben ist die Liste `[(3,4,5), (10,8,6), (25,21,12)]`, deren Einträge jeweils die Maße eines Quaders angeben (Länge, Breite und Höhe). Berechne das Volumen sowie die Oberfläche der einzelnen Quader.

Lösung:

Das Volumen eines Quaders ist gleich dem Produkt aus Länge l, Breite b und Höhe h; seine Oberfläche ist gleich der doppelten Summe aller Flächen, die sich aus je zwei der drei Größenangaben berechnen lassen:

$$V_{\text{Quader}} = h \cdot b \cdot l$$

$$A_{\text{Quader}} = 2 \cdot (h \cdot b + b \cdot l + l \cdot h)$$

In Python können die gesuchten Größen (auch ohne `sympy`) beispielsweise durch Definition von geeigneten Funktionen berechnet werden:

```
# Variablen initiieren:
cuboid_dimensions = [(3,4,5), (10,8,6), (25,21,12)]
cuboid_surfaces = []
cuboid_volumes = []

# Funktionen definieren:
def get_cuboid_surface(length, width, height):
    """ Calculate the surface of a cuboid."""

    return 2 * (height * width + width * length +  length * height)

def get_cuboid_volume(length, width, height):
    """ Calculate the volume of a cuboid."""

    return length * width * height


# Funktionen auf Liste anwenden:
for c in cuboid_dimensions:

    cuboid_surfaces.append( get_cuboid_surface(*c) )
    cuboid_volumes.append( get_cuboid_volume(*c) )

# Ergebnis:

# cuboid_surfaces: [94, 376, 2154]
# cuboid_volumes:  [60, 400, 6300]
```

In der Hauptschleife werden beide Funktionen für die jeweiligen Größenangaben aufgerufen; die Hilfsvariable `c` wird dabei, da es sich um ein um eine Sequenz handelt, mit einem davor stehenden `*` an die Funktion übergeben. Dies bewirkt, dass nicht das Zahlentripel als eigenes Objekt verwendet wird, sondern vielmehr dessen Inhalte "ausgepackt" und der Reihenfolge nach an die Funktion übergeben werden.

Diese Aufgabe lässt sich – ebenfalls wie die vorherige Aufgabe *Quaderliste – Volumen und Oberfläche* (Seite 14) mit der Python-Standardbibliothek lösen. Zu bestimmen ist das Minimum bzw. Maximimum einer Liste an Werten.

Aufgabe:

Gegeben ist folgende Liste an Quadergrößen (Länge, Breite, Höhe):

[(3,4,5),(6,8,10),(1,2,4),(12,13,32), (14,8,22),(17,3,44),(12,5,3),(10,9,11)]

Bestimme das Minumum und Maximum der Quadervolumina, die sich anhand der obigen Längenmaße ergeben.

Lösung:

Das Volumen der einzelnen Quader lässt sich als Produkt der drei Längenangaben berechnen:

$$V_{\text{Quader}} = l \cdot b \cdot h$$

```
# Variablen initiieren:
cuboid_dimensions = [(3,4,5), (6,8,10), (1,2,4), (12,13,32), (14,8,22),
    (17,3,44), (12,5,3), (10,9,11)
    ]

# Volumina berechnen:
cuboid_volumina = [ l * b * h for l,b,h in cuboid_dimensions]

# Minimum und Maximum bestimmen:
cuboid_volume_min = min(cuboid_volume_max)
cuboid_volume_max = max(cuboid_volume_max)

# Ergebnis: 8 bzw. 4992
```

Zur Erstellung der Volumina-Liste wurden hierbei eine so genannte List Comprehension genutzt. Möchte man wissen, welches Listenelement zum Wert des minimalen bzw. maximalen Volumens gehört, so kann man die Listen-Index-Funktion nutzen:

```
# Position des Minimums bzw. Maximums in der Liste bestimmen:
cuboid_volumina.index(cuboid_volume_min)
cuboid_volumina.index(cuboid_volume_min)

# Ergebnis: 2 bzw. 3
```

Python beginnt mit der Indizierung von Listenelementen bei Null, so dass die Ergebniswerte dem dritten und vierten Listenelement entsprechen.

Bei dieser einfachen Aufgabe soll anhand der Länge, Breite und Höhe eines Quaders dessen Volumen, Oberfläche und Raumdiagonale bestimmt werden.

Aufgabe:

Bestimme zu den Massen $l = 10$, $b = 8$ und $c = 6$ das Volumen, die Oberfläche und die Raumdiagonale eines Quaders.

Lösung:

Die gesuchten Größen lassen sich folgendermaßen berechnen:

$$V_{\text{Quader}} = h \cdot b \cdot l$$

$$A_{\text{Quader}} = 2 \cdot (h \cdot b + b \cdot l + l \cdot h)$$

$$d_{\text{Quader}} = \sqrt{l^2 + b^2 + h^2}$$

Die rechte Seite der letzten Gleichung entspricht dem Betrag eines dreidimensionalen Vektors.

Zur Berechnung der Quaderdiagonale kann die Funktion `sqrt()` aus dem math-Modul genutzt werden:

```
import math as m
import functools as ft

cuboid_dimensions = [10,8,6]

cuboid_volume = ft.reduce(lambda x,y: x*y , cuboid_dimensions)

cuboid_surface = lambda
cuboid_surface = 2 * (
    cuboid_dimensions[0] * cuboid_dimensions[1] +
    cuboid_dimensions[0] * cuboid_dimensions[2] +
    cuboid_dimensions[1] * cuboid_dimensions[2]
)

cuboid_diagonal = m.sqrt(
    cuboid_dimensions[0]**2 +
    cuboid_dimensions[1]**2 +
    cuboid_dimensions[2]**2
)

# Ergebnisse:

# cuboid_volume:   480
# cuboid_surface:  376
# cuboid_diagonal: 14.142135623730951
```

Bei der Berechnung des Quadervolumens wurde, um Schreibarbeit zu sparen, die Funktion `reduce()` aus dem functools-Modul verwendet. Diese führt die durch das erste Argument angegebene Funktion schrittweise von links nach rechts auf alle Elemente einer als zweites Argument übergebenen Sequenz aus; ein Aufruf von `ft.reduce(lambda x,y: x*y, [1,2,3,4,5])`. würde beispielsweise `((((1*2)*3)*4)*5)` berechnen.

Anstelle des Lambda-Ausdrucks (quasi einer Funktion ohne Namen) kann auch die Funktion `mul()` aus dem Modul operator verwendet werden. Diese wertet das Produkt aller Werte einer (als eingigem Argument übergebenen) Liste aus:

```
import operator as op

ft.reduce(op.mul, [1,2,3]) == ft.reduce(lambda x,y: x*y, [1,2,3])

# Ergebnis: True
```

Zudem könnte die Schreibarbeit durch die Definition von Funktionen entsprechend Aufgabe *Quaderliste – Volumen und Oberfläche* (Seite 14) weiter reduziert werden.

Bei sollen für einem gegebenen Punkt die Koordinaten eines neuen Punktes bestimmt werden, der sich durch Spiegelung an der x- oder y-Achse ergibt.

Aufgabe:

Eine bestimmter Punkt P soll entweder um die x- oder y-Achse gespiegelt werden. Die Koordinaten des Punktes sowie die Wahl der Spiegelungsachse sollen durch den Benutzer eingegeben werden.

Lösung:

Die Koordinaten des gespiegelten Punkts lassen sich einfach berechnen, indem die jeweilige Koordinate mit `(-1)` multipliziert wird.

Um eine Eingabe von einem Benutzer zu erhalten, kann in Python die `input()`-Funktion genutzt werden, welche die eigegebene Textzeile als String-Variable speichert. Diese wird im folgenden Beispiels schrittweise überarbeitet, indem zunächst mögliche runde Klammern mit der String-Funktion `strip` entfernt werden, der String anschließend am Kommazeichen in eine Liste zweier Strings zerlegt wird, und zuletzt eine neue Liste mittels einer List Comprehension anhand der Elemente der bestehenden Liste erstellt wird. Dieser Schritt ist notwendig, um aus den als Strings gespeicherten Zahlenwerten Float-Variablen zu erzeugen.

```
# Koordinaten des ursprünglichen Punktes einlesen:
original_point = input("Bitte die Koordinaten des Punktes eingeben \
                       [ beispielsweise (3.0, -1.5) ]: ")
axis = input("Um welche Achse soll gespiegelt werden [x oder y]? ")

# Eingabe-String in Liste von Float-Werten umwandeln:
opoint = [float(num) for num in original_point.strip('()').split(',')]

new_point = []

if axis == 'x':

    new_point.append( opoint[0] * (-1) )
    new_point.append( opoint[1] )

if axis == 'y':

    new_point.append( opoint[0] )
    new_point.append( opoint[1] * (-1) )

print("Der an der %s-Achse gespiegelte Punkt hat die \
        Koordinaten %s" % (axis, tuple(new_point)) )
```

Der obige Code zeigt nur die grundlegende Logik auf. In einem "richtigen" Programm sollte die Eingabe nach falscher Syntax hin untersucht und gegebenenfalls ein entsprechendes Exception-Handling vorgenommen werden.

Bei dieser Übungsaufgabe geht es um das Differenzieren einer quadratischen Funktion.

Aufgabe:

Der Scheitel einer Parabel ist zu bestimmen. Es ist zu entscheiden, ob die Parabel nach unten oder nach oben geöffnet ist.

Lösung:

Die Orientierung einer Parabel kann man am Koeffizienten ihres quadratischen Terms ablesen; ist dieser positiv, so ist die Parabel nach oben, andernfalls nach unten geöffnet.

Der Scheitelpunkt ist das einzige Extremum einer Parabel. Um ihn zu bestimmen, bildet man daher die erste Ableitung der quadratischen Funktion und setzt diese gleich Null. So erhält man den x-Wert des Scheitelpunktes. Den zugehörigen y-Wert erhält man, indem man den x-Wert des Scheitelpunktes in die ursprüngliche Funktionsgleichung einsetzt.

Mit sympy kann die Aufgabe folgendermaßen gelöst werden:

```
import sympy as sy

# Funktionsgleichung als String einlesen:
parable_function_input = input("Bitte die Funktionsgleichung einer Parabel \
                               eingeben (beispielsweise 3*x**2 - 5*x +7): ")

# Eingelesenen String in Sympy-Ausdruck umwandeln:
parable_function = sy.S(parable_function_input)

# Orientierung der Parabel bestimmen:
if parable_function.coeff(x, n=2) > 0:
    orientation = "up"
else:
    orientation = "down"

# Erste und zweite Ableitung bilden:
parable_function_diff_1 = sy.diff(parable_function, x, 1)
parable_function_diff_2 = sy.diff(parable_function, x, 2)

# Extremstelle bestimmen (liefert Liste an Ergebnissen):
extremes = sy.solve(sy.Eq(parable_function_diff_1, 0))

# Der gesuchte x_0-Wert ist der einzige Eintrag der Ergebnisliste:
x_0 = extremes[0]

# Zugehörigen Funktionswert bestimmen:
y_0 = parable_function.subs(x, x_0)

print("Die Orientierung der Parabel ist \"%s\", ihr Scheitel liegt bei \
      (%.2f, %.2f)." % (orientation, x_0, y_0) )
```

Bei dieser Übungsaufgabe geht es das Gleichsetzen zweier geometrischer Gleichungen.

Aufgabe:

Für eine Kreis- und eine Geradengleichung ist die Lagebeziehung (Sekante, Tangente, Passante) der beiden geometrischen Objekte zueinander zu bestimmen.

Lösung:

- Wenn eine Gerade einen Kreis in zwei Punkten schneidet, dann ist die Gerade eine Sekante.
- Wenn eine Gerade einen Kreis in einem Punkt berührt, dann ist die Gerade eine Tangente.
- Wenn die Gerade und der Kreis keinen Punkt gemeinsam haben, dann ist die Gerade eine Passante.

Mit sympy kann die Aufgabe folgendermaßen gelöst werden:

```
import sympy as sy

# Funktionsgleichungen als Strings einlesen:
circle_equation_input = input("Bitte die Funktionsgleichung eines Kreises \
                              eingeben (beispielsweise x**2 + y**2 = 9): ")
linear_equation_input = input("Bitte die Funktionsgleichung einer Geraden \
                              eingeben (beispielsweise 3*x - 2*y   = 4): ")

# Eingelesenen Strings in Sympy-Ausdrücke umwandeln:
circle_equation_elements = [sy.S(item) for item in \
                            circle_equation_input.replace("=", ",").split(",")]
linear_equation_elements = [sy.S(item) for item in \
                            linear_equation_input.replace("=", ",").split(",")]

# Gleichungssystem aus Sympy-Ausdrücken erstellen:
equations = [
    sy.Eq(*circle_equation_elements),
    sy.Eq(*linear_equation_elements)
]

# Gleichungssystem lösen
solutions = sy.solve(equations)

if len(solutions) == 2:
    print("Die Gerade ist eine Sekante.")
elif len(solutions) == 1:
    print("Die Gerade ist eine Tangente.")
else:
    print("Die Gerade ist eine Passante.")
```

Beim Erstellen des Gleichungssystems wurde bei der Übergabe der einzelnen Sympy-Elemente der Stern-Operator `*` vorangestellt, um nicht die Liste, sondern deren Inhalt an die `Eq()`-Funktion zu übergeben.