Übung 2: NumPy

Name:
Gruppe:

NumPy ist ein Akronym für „Numeric Python“ oder „Numerical Python“. Es ist ein Erweiterungsmodul für Python, das größtenteils in C geschrieben ist. Dadurch wird sichergestellt, dass die zusammengestellten mathematischen und numerischen Funktionen und Funktionalitäten die höchstmögliche Ausführungsgeschwindigkeit aufweisen. **NumPy** bereichert auch die Programmiersprache Python um leistungsstarke Datenstrukturen für effiziente Berechnungen mit großen Arrays und Matrizen.
Weitere Informationen finden Sie unter: https://numpy.org/ (https://numpy.org/)
NumPy wird nicht mit einer Standard-Python-Installation installiert.

Importieren von Numpy

Sie können das Numpy-Paket mit folgendem Befehl importieren: `import numpy as np`

(Falls **NumPy** noch nicht installiert sein sollte, können Sie es einfach per Pip installieren: 'pip install numpy'.)

Einführung in NumPy

Erstellen eines Arrays

Sie können ein **NumPy**-Array mit der `array` -Funktion erstellen. Zum Beispiel:

```
a = np.array((1, 2, 3, 4, 5)) # Erstellen eines Arrays mit einer Dimension
```

Eine Python-Liste kann verwendet werden, um ein **NumPy**-Array zu erstellen. Zum Beispiel:

```
list = [2,4,6]
b = np.array(list) # Erstellen eines Arrays aus einer Python-Liste
```

Um ein NumPy-Array aus Daten in einer Datei zu lesen, kann die Funktion `np.loadtxt` verwendet werden. Wenn die Daten aus einem Text anstelle von Zahlen bestehen, muss der richtige Datentyp (z. B. „str“ für Zeichenfolgen) angegeben werden:

```
np.loadtxt('file_name',dtype='str')
```

Arrays haben im Gegenteil zu Listen eine feste Größe. Man lässt sie nicht wie Listen mit z. B. list.append() dynamisch wachsen, sondern legt sie mit der gewünschten Größe an, z. B. voller Nullen, und modifizert dann die Einträge. Allerdings, man kann aus einem NumPy-Array auch wieder eine Python-Liste machen: `a.tolist()`

Wichtige Attribute von Arrays:

```
print(a.ndim) # Zahl der Indizes
print(a.shape) # Gestalt, z. B. (n,m) für nxm-Matrix (n:Spalte, m:Reihe)
print(a.size) # Gesamtzahl der Elemente
print(a.dtype) # Typ der Elemente

print(a.itemsize) # Größe eines Elementes in Byte
print(type(a)) # Typ von a selbst
```

Zugriff auf die Elemente eines Arrays (Indizes und Slicing)

Elemente eines Arrays kann man über Indizes ansprechen. Zum Beipiel:

```
a[3] # greift auf das 4. Element eines 1D-Arrays zu
a[0,1] # greift auf das Element in der 1. Zeile und 2. Spalte eines 2D-Arrays zu.
```

Mit der *n:m:k*-Syntax kann man Teilfelder ansprechen. Dabei ist der Startindex n enthalten, der Endindex m ist nicht mehr enthalten und k ist eine mögliche Schrittweite. Zum Beipiel:

```
a[0:5:2] # greift auf die Elemente mit den Indizes 0, 2 und 4 eines 1D-Arrays zu.
a[0:2,0:2] # greift auf die 2x2-Submatrix des 2D-Arrays mit Elementen der Indizes (0,0), (0,1), (1,0) und (1,1) zu.
```

Array neu formen und Größe ändern:

NumPy hat zwei Funktionen um Array-Formen zu verändern: `np.reshape()` und `np.resize()` . Zum Beispiel:

```
a = np.array([0,1,2,3,4,5,6,7])
b = np.reshape(a,(2,4))
```

wandelt einen Vektor mit 8 Elementen in ein Array der Form (2,4) mit 4 Reihe und 2 Spalten:

b = [[0,1,2,3],[4,5,6,7]]

Arithmetische Operationen:

- Arithmetische Operationen werden elementweise ausgeführt.

Zum Beispiel für die Arrays:

```
a=np.array([[ 1, 2],
            [ 3, 4] ])
b=np.array([[ 2, 2],
            [ 5, 5]])
```

`a + b` ergibt:

```
array([[3, 4],        [8, 9]])
```

`a * b` ergibt:

```
array([[ 2,  4],        [15, 20]])
```

`1/a` ergibt:

```
array([[ 1.        ,  0.5       ],        [ 0.33333333,  0.25      ]])
```

- Alle üblichen mathematischen Funktionen werden von NumPy so bereitgestellt, dass sie elementweise auf ein Feld wirken: `np.log(a)` , `np.sqrt(a)` , usw.
- Das innere Produkt von Vektoren und das übliche Matrixprodukt sowie die Wirkung von Matrizen auf Vektoren liefert die Funktion `np.matmul()` .Ab Python-Version 3.5 kann man für `np.matmul(x,y)` auch `x @ y` schreiben. Zum Beispiel: `a @ b` ergibt:

`array([[ 12, 12],[26, 26]]) .`

Statistische Operationen:

NumPy hat einige nützliche statistische Funktionen, um das Minimum, das Maximum, das Perzentil, die Standardabweichung, die Varianz usw. aus den gegebenen Elementen im Array zu finden. Folgenden Operationen werden am meisten durchgeführt:

`np.min()` # Minimalwert eines Arrays
`np.max()` # Maximalwert eines Arrays

`np.mean()` # Mittelwert eines Arrays
`np.median()` # Median eines Arrays
`np.std()` # Standardabweichung eines Arrays

Beispielsweise berechnet dieser Code den Durchschnitt und die Standardabweichung der ersten Zeile und der letzten Spalte eines 2D-Arrays der Form (4x3):

```
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
new_a = a.reshape(4, 3)
# Durchschnitt und Standardabweichung der ersten Zeile
mean_line = np.mean(new_a[0,:])
std_line = np.mean(new_a[0,:])
print("Mittelwert der ersten Zeile: %.2f Standardabweichung der ersten Zeile: %.2f" % (mean_line, s
td_line))
# Durchschnitt und Standardabweichung der letzten Spalte
mean_col = np.mean(new_a[:,-1])
std_col = np.mean(new_a[:,-1])
print("Mittelwert der letzten Spalte: %.2f Standardabweichung der letzten Spalte: %.2f" % (mean_co
l, std_col))
```

NumPy-Funktionen

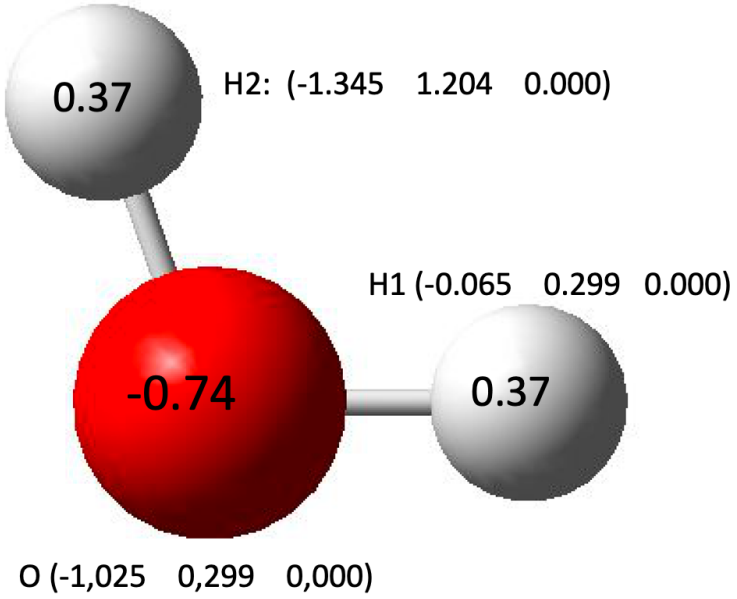
Zusätzlich zur Leistungsfähigkeit des NumPy-Arrays (auf dem einige der beeindruckendsten Bibliotheken von Python basieren) ermöglicht die NumPy-Bibliothek auch den Zugriff auf eine breite Palette nützlicher Funktionen.
`np.degrees()` #Bogenmaß in Grad umwandeln

`np.radians()` #Grad in Bogenmaß umwandeln
`np.log()` #natürlichen Logarithmus berechnen
`np.log10()` # Logarithmus zur Basis Zehn berechnen
`np.mod()` #Gibt den elementweisen Rest der Division zurück. https://numpy.org/doc/stable/reference/routines.math.html (https://numpy.org/doc/stable/reference/routines.math.html)

Aufgabe 1

a) Erstellen Sie ein mehrdimensionales NumPy-Array, das die kartesischen Koordinaten aller Atome in einem Wassermolekül und die entsprechenden Mulliken-Ladungen enthält (siehe Abbildung). [0,1 Punkte]

Jede Reihe soll dabei ein bestimmtes Atom beschreiben. Bestimmen Sie die Dimension und die Form des Arrays sowie die Anzahl der Elemente.



In [42]:
```
import numpy as np
H1 = np.array([-0.065, 0.299, 0.000, 0.37])
H2 = np.array([-1.345, 1.204, 0.000, 0.37])
O  = np.array([-1.025, 0.299, 0.000, -0.74])
water = np.array([H1, H2, O])
print(water)

print(water.size)

print(water.shape)

Form_des_arrays = type(water)
print(Form_des_arrays)

dimension = water.ndim
print(dimension)
```

```
[[-0.065  0.299  0.     0.37 ]
 [-1.345  1.204  0.     0.37 ]
 [-1.025  0.299  0.    -0.74 ]]
12
(3, 4)
<class 'numpy.ndarray'>
2
```

b) Bestimmen Sie die OH-Bindungslänge und den Abstand zwischen den Wasserstoffatomen. [0,1 Punkte]

Geben Sie dafür zunächst eine Gleichung an, mit der der Abstand zwischen zwei Punkten im dreidimensionalen Raum bestimmt werden kann.

$$d = $$

Setzen Sie dies nun unter Verwendung der Funktion `np.sqrt()` , die die Wurzel aller Elemente eines Arrays bestimmt, um.

In [61]:
```
import numpy as np
H1 = np.array([-0.065, 0.299, 0.000])
H2 = np.array([-1.345, 1.204, 0.000])
O  = np.array([-1.025, 0.299, 0.000])
#OH1=np.sqrt((H1[0]-O[0])**2)
# print(OH1)
OH2 = O - H2
print(OH2)
#np.sqrt(-0)
```

```
[ 0.32  -0.905  0.   ]
```

c) Erstellen Sie eine 3x3-Matrix der Atomkoordinaten vom Wasser. Benutzen Sie dabei die Slicing-Syntax. [0,1 Punkte]

In [3]:
```
import numpy as np
H1 = np.array([-0.065, 0.299, 0.000])
H2 = np.array([-1.345, 1.204, 0.000])
O  = np.array([-1.025, 0.299, 0.000])

wasser_matrix = np.array([H1, H2, O])
print(wasser_matrix)
```

```
[[-0.065  0.299  0.   ]
 [-1.345  1.204  0.   ]
 [-1.025  0.299  0.   ]]
```

d) Berechnen Sie den Winkel zwischen den beiden Bindungen OH1 und OH2. [0,2 Punkte]

Das Skalarprodukt zweier Vektoren **A** und **B** wird wie folgt berechnet:

$A \cdot B = |A| * |B| * \cos(\theta)$

Der Winkel zwischen zwei Vektoren ist demnach

$\theta = \arccos((A \cdot B) / (|A| * |B|))$.

Hinweis: Verwenden Sie die Numpy-Funktionen `np.dot()` und `np.arccos()` und informieren Sie sich gegebenenfalls unter https://numpy.org/doc/stable/reference/routines.math.html (https://numpy.org/doc/stable/reference/routines.math.html)

In [6]:
```
import numpy as np
H1 = np.array([-0.065, 0.299, 0.000])
H2 = np.array([-1.345, 1.204, 0.000])
O  = np.array([-1.025, 0.299, 0.000])

OH1 = H1 - O
OH2 = H2 - O

skalarprodukt = np.dot(OH1, OH2)

betrag_OH1 = np.linalg.norm(OH1)
betrag_OH2 = np.linalg.norm(OH2)

bogenmaß_OH1_OH2 = np.arccos(skalarprodukt / (betrag_OH1 * betrag_OH2))

print(f"Winkel θ in Bogenmaß: {winkel_OH1_OH2}")
```

```
Winkel θ in Bogenmaß: 1.9106668088529226
```

e) Konvertieren Sie den in Teil d) berechneten Bindungswinkel vom Bogenmaß in Grad. [0,1 Punkte]

In [5]:
```
winkelgrad = np.degrees(winkel_OH1_OH2)
print(f"Winkel θ in Grad: {winkelgrad}")
```

```
Winkel θ in Grad: 109.47314420300167
```

f) Verschieben Sie die Atomkoordinaten vom Wasser in Richtung des Vektors v=(2,3,4). [0,1 Punkte]

In [9]:
```
import numpy as np
H1 = np.array([-0.065, 0.299, 0.000])
H2 = np.array([-1.345, 1.204, 0.000])
O  = np.array([-1.025, 0.299, 0.000])

wasser_matrix = np.array([H1, H2, O])

v = np.array([2, 3, 4])

verschobene_matrix = wasser_matrix + v

print(verschobene_matrix)
```

```
[[1.935 3.299 4.   ]
 [0.655 4.204 4.   ]
 [0.975 3.299 4.   ]]
```

g) Rotieren Sie das Wassermolekül im Uhrzeigersinn um 30° um die x-Achse. [0,2 Punkte]

Definieren Sie dafür die entsprechende Rotationsmatrix. Hinweis: Eine **Rotationsmatrix** ist ein Hilfsmittel der linearen Algebra, mit dem wir etwas im euklidischen Raum drehen können, z. B. das kartesische Koordinatensystem, das wir zur Beschreibung der Positionen der Atome verwenden. Die Matrix

$$ Rx = \begin{bmatrix} 1 & 0 & 0\\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix}$$

dreht Punkte um die x-Achse gegen den Uhrzeigersinn um einen Winkel α. Um die Drehung an einem Punkt mit den Koordinaten v = (x, y, z) durchzuführen, sollte er als Spaltenvektor geschrieben und mit der Matrix Rx multipliziert werden:

$$ \begin{bmatrix} x´ \\ y´ \\ z´\end{bmatrix}= \begin{bmatrix} 1 & 0 & 0\\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Hinweis: Stellen Sie sicher, dass Sie die gedrehte Matrix wieder in das [x,y,z]-Format umwandeln. Verwenden Sie die Funktionen zur Matrixmultiplikation und zum Transponieren.

In [10]:
```
import numpy as np

H1 = np.array([-0.065, 0.299, 0.000])
H2 = np.array([-1.345, 1.204, 0.000])
O  = np.array([-1.025, 0.299, 0.000])

wasser_matrix = np.array([H1, H2, O])

a = -30
bogenmaß = np.radians(a)

# Rotationsmatrix um die x-Achse
Rx = np.array([
    [1, 0, 0],
    [0, np.cos(bogenmaß), -np.sin(bogenmaß)],
    [0, np.sin(bogenmaß), np.cos(bogenmaß)]
])

rotierte_matrix = wasser_matrix @ Rx.T

print(rotierte_matrix)
```

```
[[-0.065       0.2589416  -0.1495    ]
 [-1.345       1.04269459 -0.602     ]
 [-1.025       0.2589416  -0.1495    ]]
```

h) Zentrieren Sie die Struktur des Wassermoleküls an seinem geometrischen Zentrum. [0,2 Punkte]

Hinweis: In der Mathematik und Physik ist der Schwerpunkt, auch geometrischer Mittelpunkt einer ebenen Figur genannt, der Punkt, der durch das arithmetische Mittel aller Punkte auf der Oberfläche der Figur definiert ist. Verwenden Sie `np.mean(a,axis=0)` . Was bestimmt der Parameter `axis` ?

In [13]:
```
import numpy as np

H1 = np.array([-0.065, 0.299, 0.000])
H2 = np.array([-1.345, 1.204, 0.000])
O  = np.array([-1.025, 0.299, 0.000])

wasser_matrix = np.array([H1, H2, O])

schwerpunkt = np.mean(wasser_matrix, axis=0)

zentrierte_matrix = wasser_matrix - schwerpunkt

print(zentrierte_matrix)
print("Schwerpunkt der Matrix:", np.mean(zentrierte_matrix, axis=0))

# Der Parameter axis bestimmt,
# ob die Berechnung für Zeilen (ein Punkt der matrix), Spalten(Achse der matrix) oder der gesamten Matrix g
# axis=0 berechnet entlang der Spalte, axis=1 berechnet entlang der Zeile, axis=None berechnet über die ges
```

```
[[ 0.74666667 -0.30166667  0.        ]
 [-0.53333333  0.60333333  0.        ]
 [-0.21333333 -0.30166667  0.        ]]
Schwerpunkt der Matrix: [-7.40148683e-17  7.40148683e-17  0.00000000e+00]
```

Aufgabe 2

Die Häufigkeit der natürlichen Nickelisotope ist:

Symbol	Masse m [u]	Häufigkeit R [%]
$$ ^{58}Ni$$	57.935348	68.0769
$$ ^{60}Ni$$	59.930791	26.2231
$$ ^{61}Ni$$	60.931060	1.1399
$$ ^{62}Ni$$	61.928349	3.6345
$$ ^{64}Ni$$	63.927970	0.9256

Berechnen Sie die gewichtete durchschnittliche Masse des natürlich vorkommenden Nickels. (0,1 Punkte)

Hinweis: Jedes Atom hat seine eigene durchschnittliche Atommasse, die in der Einheit "amu" (atomic mass unit) angegeben wird. Sie wird berechnet, indem man die Summe aller Isotopenmassen nimmt und diese mit ihrer natürlichen Häufigkeit multipliziert.

$$ Atommasse = \sum\limits _{i=1} ^{n} \frac{(R[i])}{100} \cdot (m[i]) $$

In [15]:
```
import numpy as np

m = np.array([57.935348, 59.930791, 60.931060, 61.928349, 63.927970])

häufigkeit = np.array([68.0769, 26.2231, 1.1399, 3.6345, 0.9256])

atommasse = np.sum((häufigkeit / 100) * m)

print(f"Die gewichtete durchschnittliche Atommasse des Nickels beträgt: {atommasse:.6f} u")
```

```
Die gewichtete durchschnittliche Atommasse des Nickels beträgt: 58.693356 u
```

Aufgabe 3

Berechnen Sie den pH-Wert von vier H_2SO_4-Lösungen mit folgenden Konzentrationen: 0,005 M, 0,010 M, 0,015 M und 0,02 M. Erzeugen Sie ein Array, das die Konzentrationsangabe und den entsprechenden pH-Wert in zwei Spalten zusammenfasst. [0,1 Punkte]

Geben Sie zuerst die Gleichung für den pH-Wert einer starken Säure an.

$$pH = -log[H+] = -log_{10}[c]$$

Hinweis: Verwenden Sie die Funktion `np.column_stack()` . Diese Funktion bildet ein 2-D Array aus eine Folge von 1-D Arrays die als Tuple geschrieben werden könnten.

In [18]:
```
import numpy as np

c = np.array([0.005, 0.010, 0.015, 0.02])

pH_werte = -np.log10(c)

zusammenfassung = np.column_stack((c, pH_werte))

print("Tabelle der Konzentrationen (c) und den zugehörigen pH-Werten:")
print(zusammenfassung)
```

```
Tabelle der Konzentrationen (c) und den zugehörigen pH-Werten:
[[0.005      2.30103   ]
 [0.01       2.        ]
 [0.015      1.82390874]
 [0.02       1.69897   ]]
```

Aufgabe 4

Es wurde ein Experiment zehnmal durchgeführt, um einen möglichst genauen Wert des Schmelzpunktes von Wasser zu erhalten.

Experiment	Schmelzpunkt [°C]
1	98.5
2	99.9
3	100.6
4	99.3
5	100.7
6	99.4
7	98.4
8	99.5
9	99.3
10	100.7

a) Ermitteln Sie das Minimum, das Maximum und die Standardabweichung der Messwerte. [0,1 Punkte]

In [19]:
```
import numpy as np

schmelzpunkte = np.array([98.5, 99.9, 100.6, 99.3, 100.7, 99.4, 98.4, 99.5, 99.3, 100.7])

minimum = np.min(schmelzpunkte)
maximum = np.max(schmelzpunkte)
standardabweichung = np.std(schmelzpunkte)

print(f"Minimum des Schmelzpunkts: {minimum:.2f} °C")
print(f"Maximum des Schmelzpunkts: {maximum:.2f} °C")
print(f"Standardabweichung des Schmelzpunkts: {standardabweichung:.2f} °C")
```

```
Minimum des Schmelzpunkts: 98.40 °C
Maximum des Schmelzpunkts: 100.70 °C
Standardabweichung des Schmelzpunkts: 0.80 °C
```

b) Es wurde festgestellt, dass die Kalibrierung des Messinstrumentes fehlerhaft war. Addieren Sie 0,2 °C zu allen Messwerten. Konvertieren Sie die Werte in eine Zeichenkette und geben Sie das Array aus. [0,1 Punkte]

In [1]:
```
import numpy as np

schmelzpunkte = np.array([98.5, 99.9, 100.6, 99.3, 100.7, 99.4, 98.4, 99.5, 99.3, 100.7])
schmelzpunkte_neu = schmelzpunkte + 0.2
zeichenkette = schmelzpunkte_neu.astype(str)

print("Korrigierte Schmelzpunkte als Zeichenketten-Array:")
print(zeichenkette)
```

```
Korrigierte Schmelzpunkte als Zeichenketten-Array:
['98.7' '100.10000000000001' '100.8' '99.5' '100.9' '99.60000000000001'
 '98.60000000000001' '99.7' '99.5' '100.9']
```

Aufgabe 5

a) Es wurden für 20 Tage Temperaturen in Fahrenheit gemessen, die in der Datei temp.txt gespeichert wurden. Verwenden Sie numpy, um diese Datei als Array zu laden. Geben Sie die Temperaturen absteigend sortiert aus. [0,1 Punkte]

Hinweis: Verwenden Sie die Funktion `np.sort()` . Sie können die Syntax `array[::-1]` verwenden, um das Array umzukehren.

In [5]:
```
import numpy as np

temp = np.loadtxt("temp.txt")

temp_sort = np.sort(temp)[::-1]

print("Temperaturen in Fahrenheit:")
print(temp_sort)
```

```
Temperaturen in Fahrenheit:
[189.  98.  97.  88.  81.  79.  73.  66.  62.  60.  57.  55.  50.  49.
  40.  38.  35.  25.  24.  20.]
```

b) Schreiben Sie eine Funktion, die Temperaturen in °F übergeben bekommt und diese in °C zurückgibt. Schreiben Sie dann eine weitere Funktion, die eine Temperatur von °C in K umrechnet. [0,1 Punkt]

In [12]:
```
import numpy as np

temp = np.loadtxt("temp.txt")

temp_sort = np.sort(temp)[::-1]

def fahrenheit_zu_celsius(fahrenheit):
    celsius = (fahrenheit -32) * (5/9)
    return celsius

def celsius_zu_kelvin(celsius):
    kelvin = celsius + 273
    return kelvin

T_in_Celsius = fahrenheit_zu_celsius(temp_sort)
print(f"{T_in_Celsius}")

T_in_Kelvin = celsius_zu_kelvin(T_in_Celsius)
print(f"{T_in_Kelvin}")
```

```
[87.22222222 36.66666667 36.11111111 31.11111111 27.22222222 26.11111111
 22.77777778 18.88888889 16.66666667 15.55555556 13.88888889 12.77777778
 10.          9.44444444  4.44444444  3.33333333  1.66666667 -3.88888889
 -4.44444444 -6.66666667]
[360.22222222 309.66666667 309.11111111 304.11111111 300.22222222
 299.11111111 295.77777778 291.88888889 289.66666667 288.55555556
 286.88888889 285.77777778 283.         282.44444444 277.44444444
 276.33333333 274.66666667 269.11111111 268.55555556 266.33333333]
```

c) Verwenden Sie die Funktionen, um die gemessenen Temperaturen in °C und in K auszugeben. [0,1 Punkte]

In [13]:
```
import numpy as np

temp = np.loadtxt("temp.txt")

temp_sort = np.sort(temp)[::-1]

def fahrenheit_zu_celsius(fahrenheit):
    celsius = (fahrenheit -32) * (5/9)
    return celsius

def celsius_zu_kelvin(celsius):
    kelvin = celsius + 273
    return kelvin

T_in_Celsius = fahrenheit_zu_celsius(temp_sort)
print(f"{T_in_Celsius}")

T_in_Kelvin = celsius_zu_kelvin(T_in_Celsius)
print(f"{T_in_Kelvin}")
```

```
[87.22222222 36.66666667 36.11111111 31.11111111 27.22222222 26.11111111
 22.77777778 18.88888889 16.66666667 15.55555556 13.88888889 12.77777778
 10.          9.44444444  4.44444444  3.33333333  1.66666667 -3.88888889
 -4.44444444 -6.66666667]
[360.22222222 309.66666667 309.11111111 304.11111111 300.22222222
 299.11111111 295.77777778 291.88888889 289.66666667 288.55555556
 286.88888889 285.77777778 283.         282.44444444 277.44444444
 276.33333333 274.66666667 269.11111111 268.55555556 266.33333333]
```

Aufgabe 6

In der Datei p53.csv sind Inhibitoren des Tumorsuppressorproteins p53 aufgeführt. Es handelt sich um eine Tabelle mit der ID der Verbindung und ihrem Molekulargewicht.

Laden Sie unter Verwendung von numpy diese Datei und bestimmen Sie die Anzahl an Verbindungen mit einem Molekulargewicht unter 1500 Da. Geben Sie zusätzlich die IDs der Verbindungen aus, deren Molekulargewicht unter 800 Da liegt. [0,2 Punkte]

Hinweis: In einer csv-Datei sind die Werte durch Kommata (,) getrennt. Um diese Datei in ein multidimensionales Array zu laden, verwenden Sie

das Attribut delimiter in `np.loadtxt('filename', delimiter=',')` . Verwenden Sie if- und elif-Anweisungen.

In [15]:
```
import numpy as np

data = np.loadtxt('p53.csv', delimiter=',', dtype=float)

ids = data[:, 0]
molekulargewichte = data[:, 1]

unter_1500 = np.sum(molekulargewichte < 1500)

ids_unter_800 = ids[molekulargewichte < 800]

print(f"Anzahl der Verbindungen mit Molekulargewicht unter 1500 Da: {unter_1500}")
print("IDs der Verbindungen mit Molekulargewicht unter 800 Da:", ids_unter_800)
```

```
Anzahl der Verbindungen mit Molekulargewicht unter 1500 Da: 8
IDs der Verbindungen mit Molekulargewicht unter 800 Da: [5079127. 1615784. 3883378. 1797020.]
```

In [ ]: