Python Crashkurs – Part 3

Mentoring – WiSe 25/26

Patricia Gerbig
Freie Universität Berlin
Fachbereich Mathematik und Informatik

06. Oktober 2025

Webseite mit Material

Aufgabe -1

1. Schreibe eine function bintodez(d), die eine Binärzahl (z.B. als String) übergeben bekommt, und diese in die Dezimaldarstellung umwandelt.
2. Extra: Schreibe eine Funktion deztobin(d), die eine Dezimalzahl d übergeben bekommt und in eine binärzahl umrechnet.

Inhalt

Inhalt

List-Comprehension

Mit **List-Comprehension** kann aus einer bereits bestehenden Liste eine neue erstellt werden

```
newlist = [Ausdruck for item in list if condition == True
    ]
```

Beispiel:

```
list1 = list(range(1,21))
newlist1 = [x for x in list1 if x % 2 == 0]
print(newlist1)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Aufgabe 0

1. Wir haben eine Liste von Namen:

```
1  namen = ['Lisa', 'Benjamin', 'Emil', 'Stella', 'Nico',
       'Moritz', 'Tara']
```

   Filtert alle Namen die ein a enthalten in eine neue Liste.

2. Recherschiert was wie man zufällige Zahlen in Python ausgibt. Schreibt eine Funktion lotto() die einem 6 Zufallszahlen von 1 bis 49 ausgibt. (Ihr könnt dabei gerne ignorieren, dass sich die Zahlen nicht doppeln dürfen)

Inhalt

NumPy importieren

- Um *NumPy* nutzen zu können, ist folgende Zeile notwendig:

```
1  import numpy as np
```

Source Code 1: Numpy importieren

- Sollte *beim importieren* eine Fehlermeldung auftreten, ist *NumPy* vermutlich nicht installiert. *NumPy* lässt sich meistens mit dem Befehl `pip3 install numpy` installieren.

Inhalt

Matrix

- Eine $(n \times m)$-Matrix ist eine rechteckige Anordnung von Zahlen in m Spalten und n Zeilen:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \vdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}$$

- Beispiele:

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Eine (1×1)-Matrix entspricht einer Zahl. $(1 \times m)$- und $(n \times 1)$-Matrizen werden auch *Vektoren* genannt. Vektoren wird in der Regel ein Kleinbuchstabe zugewiesen (siehe Beispiel).

Matrix erstellen

- Eine Matrix wird mit dem Befehl `np.array(list)` erstellt, wobei `list` eine Liste ist, welche die Dimension und die Einträge einer Matrix vorgibt.
- Beispiele:

```
1  # Erzeugt eine (3 x 4)-Matrix
2  A = np.array([[3, 2, 1, 0],
3                [2, 1, 0, 0],
4                [1, 0, 0, 0]])
5  # Erzeugt einen Vektor ((1 x 3)-Matrix) mit 3 Einträgen
6  x = np.array([1, 2, 3])
```

Source Code 2: Wie man eine Matrix erstellt

Aufgabe 1

Erstelle folgende Matrizen:

$$x = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \end{pmatrix} \qquad A = \begin{pmatrix} 1 & -2 & 5 \\ -2 & 7 & 3 \end{pmatrix}$$

Eigenschaften einer Matrix

- Sei A eine Matrix.

Eigenschaften einer Matrix	
`A.dtype`	Datentyp der Elemente
`A.ndim`	Dimension der Matrix
`A.shape`	Gestalt der Matrix
`A.size`	Anzahl der Elemente
`A.shape[n]`	Anzahl der Elemente für die n-te Dimension

Table 1: Eigenschaften einer Matrix

Beispiele:

```
1 A = np.array([[1, 2, 3],
2                [4, 5, 6]])
3 A.dtype # dtype('int32')
4 A.ndim # 2
5 A.shape # (2, 3)
6 A.shape[0] # 2 Zeilen
7 A.shape[1] # 3 Spalten
8 A.size # 6
```

Source Code 3: Eigenschaften einer Matrix

Matrix - Operationen

- Für Matrizen gibt es eine Menge von Rechenoperationen.
- Matrizenaddition:

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} + \begin{pmatrix} 0 & 2 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+2 \\ 2+1 & 4+3 \end{pmatrix} = \begin{pmatrix} 1 & 5 \\ 3 & 7 \end{pmatrix}$$

- Matrizensubtraktion:

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} - \begin{pmatrix} 0 & 2 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 1-0 & 3-2 \\ 2-1 & 4-3 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Matrix - Operationen

- Skalarmultiplikation:

$$\pi \cdot \begin{pmatrix} +1 & 0 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} \pi \cdot (+1) & \pi \cdot 0 \\ \pi \cdot 2 & \pi \cdot 1 \end{pmatrix} = \begin{pmatrix} +\pi & 0 \\ 2\pi & \pi \end{pmatrix}$$

- Matrixmultiplikation:

$$\begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 \cdot 1 + 2 \cdot 1 & 3 \cdot 1 + 2 \cdot 0 \\ 2 \cdot 1 + 1 \cdot 1 & 2 \cdot 1 + 1 \cdot 0 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$$

- Transposition:

$$\begin{pmatrix} -2 & 3 \\ -1 & 4 \\ 0 & 5 \end{pmatrix}^T = \begin{pmatrix} -2 & -1 & 0 \\ 3 & 4 & 5 \end{pmatrix}$$

Matrix-Operationen

- Sei A eine $(n \times m)$-Matrix und B eine $(k \times l)$-Matrix. Sei c eine beliebige Zahl.

Matrixoperationen (unär)	
`A.T`	Transponiert die Matrix A
Matrixoperationen (binär)	
`A + B`	Matrizenaddition ($n = k$ und $m = l$)
`A - B`	Matrizensubtraktion ($n = k$ und $m = l$)
`A * B`	Komponentenweise Multiplikation ($n = k$ und $m = l$)
`c * A`	Skalarmultiplikation
`A @ B`	Matrizenmultiplikation (m = k)
`A / B`	Komponentenweise Division ($n = k$ und $m = l$)

Table 2: Rechenoperationen auf Matrizen

Matrix-Operationen


```python
A = np.array([[1, 2, 3],
              [4, 5, 6]])
B = np.array([[0, 1, -1],
              [1, 0, 2]])
A + B
A - B
A * B # Komponentenweise Multiplikation
A / B # Komponentenweise Division
# Was ist das Ergebnis von A @ B?
# Sind die Ergebnisse unten alle gleich?
A @ B.T
B @ A.T
A.T @ B
B.T @ A
```

Source Code 4: Matrix-Operationen

Aufgabe 2

Schreibe eine Funktion *is_invertible(A)*, die überprüft ob eine Matrix invertierbar ist.
Eine Matrix A ist invertierbar, wenn folgende bedingungen gelten:

- A is quadratisch
- Die Determinante von A is ungleich 0.

Einheitsmatrix I_n

- Dimension n - ist $(n \times n)$-Matrix
- Es gilt:

$$A \cdot I_n = A = I_n \cdot A$$

für alle $(n \times n)$-Matrizen A.

- Beispiel für $n = 2$:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

- Eine solche Matrix lässt sich mit `np.eye(x)` erstellen, wobei x vom Typ `int` ist:

```
1 >>> np.eye(2)
2 array([[1., 0.],
3        [0., 1.]])
```

Source Code 5: Beispiel für `np.eye(x)`

Nützliche Matrizen

- Matrix mit Einsen an jeder Stelle: `np.ones(x)`.
- `x` ist ein Tupel oder nichtnegativer `int` - gibt die Dimension an.

```
>>> np.ones(3)
array([1., 1., 1.])
>>> np.ones((2, 2))
array([[1., 1.],
       [1., 1.]])
```

Source Code 6: Beispiel für `np.ones(x)`

- Matrix mit einer Null an jeder Stelle erstellt: `np.zeros(x)`.

```
>>> np.zeros(3)
array([0., 0., 0.])
>>> np.zeros((2, 3))
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Source Code 7: Beispiel für `np.zeros(x)`

Aufgabe 3

Erstelle folgende (7×7)-Matrix - ohne diese ganzen Zahlen einzutippen:

$$\begin{pmatrix} 3 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 3 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 3 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 3 \end{pmatrix}$$

Gitter erstellen

- In der Numerik wird oft ein *äquidistantes Gitter* benötigt: Für ein Intervall $I \subseteq [a, b] \neq \emptyset$ und $n \in \mathbb{N}$ ist das die Menge

$$\{a + k \cdot h \mid k = 0, 1, \ldots, n\}$$

  mit *Schrittweite* $h = (b + a)/n$.
- Für diesen Zweck gibt es zwei verschiedene Befehle: `np.arange(...)` und `np.linspace(...)`.
- Beispiel für `np.arange(...)`:

```
1 >>> np.arange(6) # Die 6 selbst ist exklusiv!
2 array([0, 1, 2, 3, 4, 5])
3 >>> np.arange(2, 8)
4 array([2, 3, 4, 5, 6, 7])
5 # Für nichtganzzahlige Schrittweiten np.linspace(...)
6 # besser verwenden.
7 >>> np.arange(1, 3, 0.25)
8 array([1.  , 1.25, 1.5 , 1.75, 2.  , 2.25, 2.5 , 2.75])
```

Source Code 8: Beispiel für `np.arange(...)`

Gitter erstellen

- Beispiel für *np.linspace(...)*:

```
>>> np.linspace(1, 3, 6)
array([1. , 1.4, 1.8, 2.2, 2.6, 3. ])
# Ohne drittes Argument werden 50 Gitterpunkte
# generiert
>>> np.linspace(1, 50)
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.,
       10., 11., 12., 13., 14., 15., 16., 17., 18.,
       19., 20., 21., 22., 23., 24., 25., 26., 27.,
       28., 29., 30., 31., 32., 33., 34., 35., 36.,
       37., 38., 39., 40., 41., 42., 43., 44., 45.,
       46., 47., 48., 49., 50.])
```

Source Code 9: Beispiel für *np.linspace()*

Indizierung von Matrizen

- An dieser Stelle gehen wir von einem Vektor aus ($= (1 \times m)$- oder $(n \times 1)$-Matrix).

Indizierung von Vektoren	
`[i]`	i-ter Eintrag
`[:i]`	Teilvektor bis Position $(i+1)$
`[i:]`	Teilvektor ab Position i
`[i:j]`	Teilvektor ab Pos. i bis Pos. $(j+1)$
`[i:j:k]`	Jeder k-te Eintrag ab Pos. i bis Pos. $(j+1)$

Table 3: Zugriff auf Elemente

Beispiele:

```
1  a = np.array([2, 1, 3, 4,
       6, 5, 8, 7])
2  a[0] # Erstes Element
3  a[-1] # Letztes Element
4  a[5:] # [5, 8, 7]
5  a[:2] # [2, 1]
6  a[1::3] # [1, 6, 7]
7  a[2:6:2] # [3, 6]
8  a[:4:2] # [2, 3]
9  a[::3] # [2, 4, 8]
10 a[::-1] # a rückwärts
```

Source Code 10: Zugriff auf Elemente

Indizierung von Matrizen

- Nun nehmen wir eine $(n \times m)$-Matrix mit $n, m \geq 2$ an.

Indizierung von Matrizen	
`[i, j]`	Eintrag in i-ter Zeile und j-ter Spalte
`[i, :]`	Ganze i-te Zeile
`[:, j]`	Ganze j-te Spalte
`[:, :]`	Kopie einer Matrix

Table 4: Zugriff auf Elemente

Beispiele:

```
1 A = np.array([[1, 2, 3],
2                [4, 5, 6],
3                [7, 8, 9]])
4 # Eintrag in 3.ter Zeile,
5 # 2.ter Spalte
6 A[2, 1]
7 A[1, :] # Zweite Zeile
8 A[:, 0] # Erste Spalte
```

Source Code 11: Zugriff auf Elemente

Aufgabe 4

Schreibe eine Funktion, die alle Werte innerhalb einer Matrix multipliziert. Beispiel:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Die Funktion soll dann $1 \cdot 2 \cdot 3 \cdot \ldots$ berechnen.

Inverse einer Matrix – Berechnung

- In *NumPy* lässt sich eine Matrix A mit `numpy.linalg.inv(A)` invertieren.

```
>>> A = np.array([[3, -1], [-2, 1]])
>>> B = np.linalg.inv(A) # Matrix wird invertiert!
>>> A
array([[ 3, -1],
       [-2,  1]])
>>> B
array([[1., 1.],
       [2., 3.]])
>>> A @ B
array([[ 1.0000000e+00, -4.4408921e-16],
       [ 0.0000000e+00,  1.0000000e+00]])
>>> B @ A
array([[ 1.0000000e+00,  0.0000000e+00],
       [-8.8817842e-16,  1.0000000e+00]])
```

Source Code 12: Invertieren einer Matrix

Lineare Gleichungssysteme

- Mit Matrizen lassen sich *lineare Gleichungssysteme* beschreiben.
- Beispiel: Das lineare Gleichungssystem

$$\begin{aligned} 4 \cdot x + 2 \cdot y &= 1 \\ x + y &= 1 \end{aligned}$$

lässt sich folgendermaßen mit einer Matrix beschreiben:

$$\begin{pmatrix} 4 & 2 \\ 1 & +1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Multiplizieren wir beide Seiten mit der inversen Matrix, erhalten wir

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ 1 & +1 \end{pmatrix}^{+1} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1/6 & 2/6 \\ 1/6 & +4/6 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1/2 \\ +1/2 \end{pmatrix}$$

Lineare Gleichungssysteme

- Lineare Gleichungssysteme mittels *np.linalg.inv(A)* zu lösen ist in der Praxis nicht empfehlenswert. Dafür verwendet man

  np.linalg.solve(A, b)

- Beispiel:

```
1  >>> A = np.array([[4, 2], [1, -1]])
2  >>> b = np.ones(2)
3  >>> x = np.linalg.solve(A, b)
4  >>> x
5  array([ 0.5, -0.5])
6  >>> A @ x
7  array([1., 1.])
```

Source Code 13: Lösen eines LGS

Inhalt

Mathematische Konstanten

Die Definition von π und e sind in *Math* und *NumPy* gleich.

Konstanten	
`np.e`	Eulersche Zahl
`np.pi`	Kreiszahl

Table 5: In *Numpy* definierte Konstanten

```
1 >>> np.e
2 2.718281828459045
3 >>> np.pi
4 3.141592653589793
```

Source Code 14: Konstanten

Trigonometrische Funktionen

- Im Gegensatz zu *Math*, können die trigonometische Funktionen in *NumPy* auf einzelne Werte und auf Matrizen angewandt werden.
- Die Funktionen werden bei Matrizen *komponentenweise* angewandt.

Trigonometrische Funktionen	
`np.sin(A)`	Sinus
`np.cos(A)`	Cosinus
`np.tan(A)`	Tangens
`np.arcsin(A)`	Arkussinus
`np.arccos(A)`	Arkuscosinus
`np.arctan(A)`	Arkustangens

Table 6: Trigonometrische Funktionen

```
1 A = np.array([[0.25,
      0.5], [0.75, 1]])
2 np.sin(A)
3 np.cos(A)
4 np.tan(A)
5 np.arcsin(A)
6 np.arccos(A)
7 np.arctan(A)
```

Source Code 15: Funktionen

Weitere Funktionen

Exponentialfunktion und Logarithmen	
`np.exp(A)`	Exponentialfunktion
`np.log(A)`	Natürlicher Logarithmus
`np.log2(A)`	Dualer Logarithmus
`np.log10(A)`	Dekadischer Logarithmus

Weitere Funktionen	
`np.sqrt(A)`	Quadratwurzel
`np.ceil(A)`	Aufrunden
`np.floor(A)`	Abrunden
`np.degrees(A)`	Wandelt Radiant in Grad für jeden Eintrag um
`np.radians(A)`	Wandelt Grad in Radiant für jeden Eintrag um
`A.cumsum()`	Bildet die *kumulative Summe* der Einträge

Table 7: Weitere Funktionen aus *NumPy*