Numerisches Python mit NumPy

Proseminar Python

Nico Hädicke

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

28-06-2022

Gliederung

Einleitung

- NumPy installieren

```
pip install numpy
```

```
pip3 install numpy
```

- NumPy importieren

```
import numpy as np
```

- Nur ein kleiner Einblick

Arrays

- Was ist ein Array?
  - Ansammlung von Werten
  - Selber Datentyp üblich
  - Dimension und Form
- Array erstellen

```
1 arr = np.array([1, 2, 3])
```

dtype

- Was ist np.dtype?
  - Datentyp des gesamten Arrays
  - VisibleDeprecationWarning bei Mischung

```
1 arr = np.array([1, 2, [3]], dtype=object)
```

- Auswirkung
  - Bytes/Element festgelegt
  - Verschwendung bei Mischung
- Verbindung zu C
  - Zusammenhängender C Array
  - Elemente für Python verpackt

```
1 a = np.array([1, 1])
2 b = np.array([1, [1]])
3 print(a[0] is a[0])  # Output: False
4 print(b[0] is b[0])  # Output: True
```

Speicherplatz (Listen)

- Ganze Liste: $54 + 8 * len(lst) + len(lst) * 28$

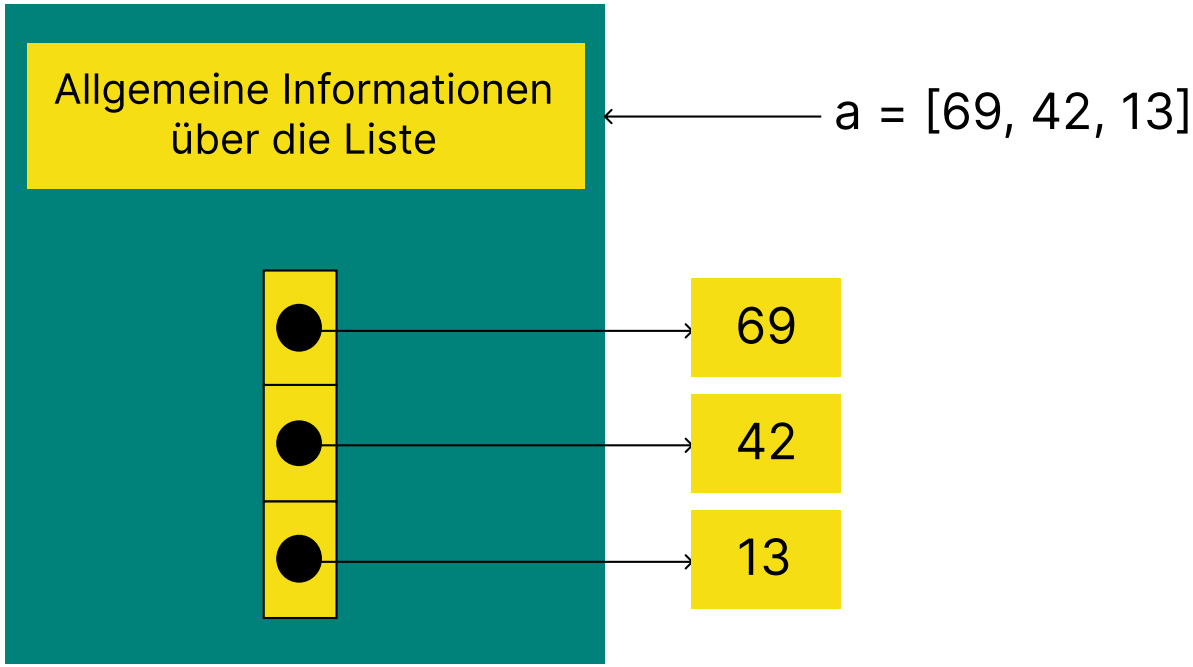

Abbildung: Darstellung der Listen-Speicherung [1]

- Hier: $54 + 8 \cdot 3 + 3 \cdot 28 = 164$ Bytes
- Tatsächlich: 204 Bytes

Speicherplatz (Arrays)

- Element: 4 Byte pro Int
- Ganzes Array: $112 + len(arr) \cdot 4$ Bytes

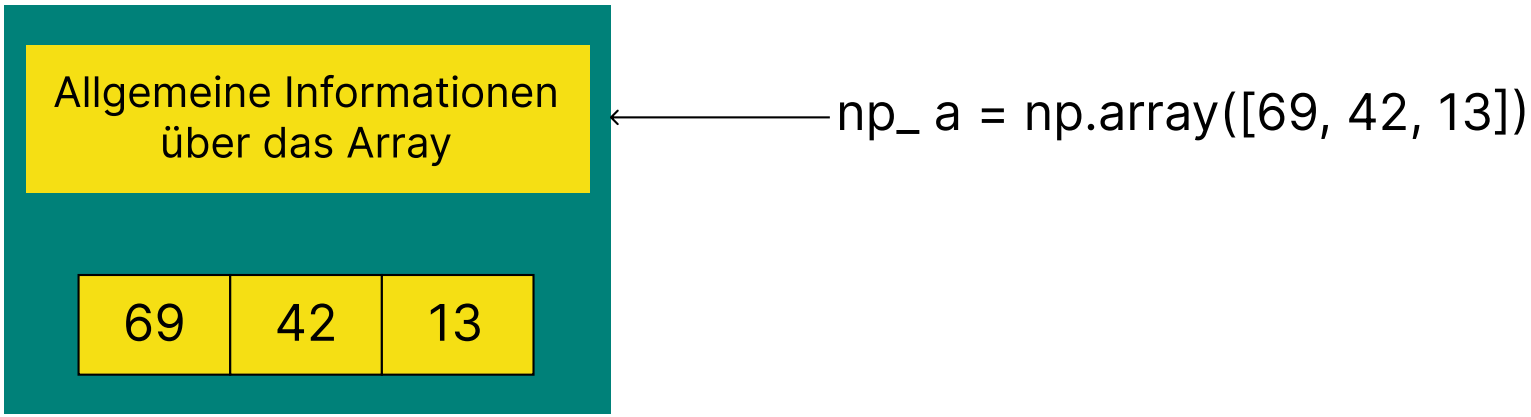

Abbildung: Darstellung der Array-Speicherung [1]

- Hier: $112 + 3 \cdot 4 = 124$ Bytes
- Keine Reservierung

Geschwindigkeit

- Parallele Ausführung von Operationen

```
1 list1 = [i for i in range(10000000)]
2 list2 = [i for i in range(10000000)]
3 list1 + list2
```

```
1 np_1 = np.array(list1)
2 np_2 = np.array(list2)
3 np_1 + np_2
```

- NumPy ist in diesem Fall 13x schneller

Arithmetischen Operationen

- Limitationen von normalen Arrays
  - Meisten arithmetischen Operationen funktionieren nicht

```
1 a = [1, 2]
2 b = [3, 4]
3 print(a + b)  # Output: [1, 2, 3, 4]
```

```
1 np_a = np.array([1, 2])
2 np_b = np.array([3, 4])
3 print(a + b)  # Output: [4, 6]
```

Multidimensionalität und Indexing

- Ähnliche Anwendung in beiden Fällen

```
1 a = [[1, 2], [3, 4]]
2 np_a = np.array(a)
3 print(a[1])  # Output: [3, 4]
4 print(np_a[1])  # Output: [3, 4]
```

Slicing

- Aufbau wird beibehalten
- von:bis:sprung

```
1 a = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
2
3 print(a[:1, :, ::2])  # Output: [[[1], [3]]]
```

Iterieren

```
1 a = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

- Einfache for-Schleifen Verschachtelung

```
1 for x in a:
2     for y in x:
3         for z in y:
4             print(z) # Output: 1 -> 2 -> 3 ...
```

- nditer

```
1 for x in np.nditer(a[:, 1::2, -1]):
2     print(x) # Output: 4 -> 8
```

Was ist Broadcasting?

- Regelset zum Umgehen mit Arrays unterschiedlicher Form

```
1 a = np.array([1, 2, 3])
2 b = np.array([2])
3
4 print(a * b)  # Output: [2, 4, 6]
```

- Array nur für Berechnung "verlängert"
- Kein extra Speicherplatz benötigt

Genereller Ablauf

- Zwei Arrays werden dimensionsweise verglichen
- Zwei Dimensionen sind kompatibel wenn:
  1. Sie gleich groß sind
  2. Eine der Dimensionen eine Größe von 1 hat
- Sonst ValueError

Shape

- Gibt Anzahl Elemente jeder Dimension zurück

```
1 arr = np.array([1, 2, 3], ndmin=6)
2
3 print(arr)  # Output: [[[[[[1, 2, 3]]]]]]
4 print(arr.shape)  # Output: (1, 1, 1, 1, 1, 3)
```

- Komplexeres Beispiel

```
1 arr = np.array([[[1, 2]], [[3, 4]], [[5, 6]]])
2
3 print(arr.shape)  # Output: (3, 1, 2)
```

Broadcast Beispiele

- Kein Code!

```
1 A         (4d array): 8 x 1 x 6 x 1
2 B         (3d array):     7 x 1 x 5
3 Ergebnis  (4d array): 8 x 7 x 6 x 5
```

- Anderes Beispiel:

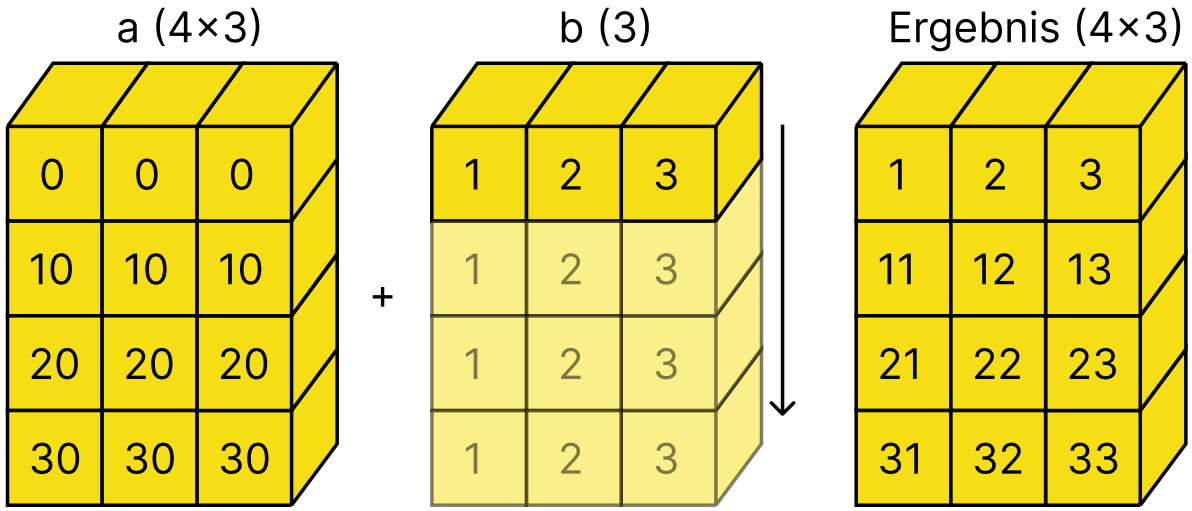

Abbildung: Visualisierung von Broadcasting [4]

Was sind ufuncs?

- Steht für Universal Functions
- Werden angewandt auf ndarrays
- Zum Implementieren von "vectorization"
  - Ersetzen von Iteration
  - Schneller aufgrund von parallelem Abarbeiten
  - Nicht pauschal besser
- Addieren zweier Listen
  - 100 Mio Elemente/Liste - 2x schneller

Beispiel

- Addieren von Elementen gleichdimensionaler Listen

```
1 x = [1, 2, 3, 4]
2 y = [4, 5, 6, 7]
3 z = []
```

- Mit for-Schleife

```
1 for i in range(4):
2     z.append(x[i] + y[i])
3 print(z)  # Output: [5, 7, 9, 11]
```

- Mit ufunc add(x, y)

```
1 print(np.add(x, y))  # Output: [5, 7, 9, 11]
```

Eigene ufuncs erstellen

1. Eigene Funktion erstellen

```
1 def mysub(x, y):
2     return x-y
```

2. Mit frompyfunc(funktion, inputs, outputs) zur NumPy ufunc library hinzufügen

```
1 mysub = np.frompyfunc(mysub, 2, 1)
```

3. Fertig

```
1 print(mysub([1, 2, 3, 4], [5, 6, 7, 8]))  #
    ↪ Output: [-4, -4, -4, -4]
```

Verschiedene ufuncs

- Einfache Arithmetik
  - Nur sinnvoll für nicht NumPy Arrays
- Entfernen von Nachkommastellen
- Runden auf gewünschte Nachkommastelle
- Finden von KGV und GGT
- Trigonometrie
- Hyperbolische Geometrie
  - Bekannte Geometrie ohne Parallelenaxiom

Zusammenfassung

- NumPy Arrays
  - Spart Speicher
  - Höhere Geschwindigkeit
  - Verschiedene mitgelieferte Funktionen
- Broadcasting
  - Vergleich zweier Arrays dimensionsweise
  - Dimensionen angepasst ohne Speicherverbrauch
- ufuncs
  - Ersetzen von Iteration
  - Eigene Methoden
  - Funktionen für sehr komplexe Mathematik

Literatur

1. https://webcourses.ucf.edu/courses/1249560/pages/python-lists-vs-NumPy-arrays-what-is-the-difference
2. https://www.w3schools.com/python/NumPy/
3. https://NumPy.org/doc/stable/user/absolute_beginners.html
4. https://NumPy.org/doc/stable/user/basics.broadcasting.html
5. https://numpy.org/devdocs/reference/arrays.dtypes.html
6. https://numpy.org/doc/stable/user/basics.types.html
7. https://stackoverflow.com/questions/41325427/numpy-ufuncs-speed-vs-for-loop-speed
8. https://stackoverflow.com/questions/40911491/some-confusions-on-how-numpy-array-stored-in-python