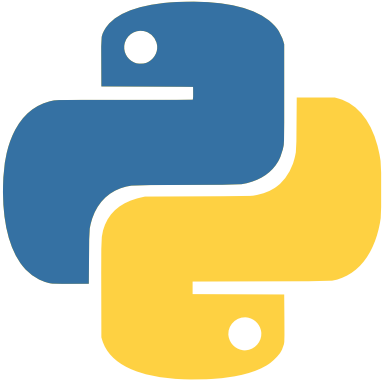
Universität Stuttgart
Projekt digit@L – BOOST. SKILLS. SUPPORT.



Programmierkurs Python

NumPy: ndarrays

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

CC BY

NumPy: ndarrays – Die zentrale Datenstruktur

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar
- In NumPy: `ndarray` für n-dimensionale Arrays

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar
- In NumPy: `ndarray` für n-dimensionale Arrays
  - Vektoren (1D), Matrizen (2D), Tensoren (nD)

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar
- In NumPy: `ndarray` für n-dimensionale Arrays
  - Vektoren (1D), Matrizen (2D), Tensoren (nD)
- Limitierung der Funktionalität steigert **Effizienz**

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar
- In NumPy: `ndarray` für n-dimensionale Arrays
  - Vektoren (1D), Matrizen (2D), Tensoren (nD)
- Limitierung der Funktionalität steigert **Effizienz**
  - Speicherplatz komplett bei Anlegung reserviert

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar
- In NumPy: `ndarray` für n-dimensionale Arrays
  - Vektoren (1D), Matrizen (2D), Tensoren (nD)
- Limitierung der Funktionalität steigert **Effizienz**
  - Speicherplatz komplett bei Anlegung reserviert
  - b Bytes pro Eintrag, konsekutiv (linear) im Arbeitsspeicher

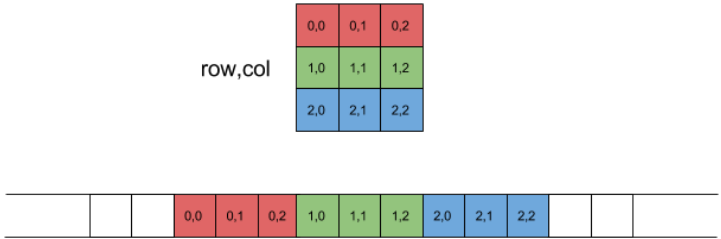
ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar
- In NumPy: `ndarray` für n-dimensionale Arrays
  - Vektoren (1D), Matrizen (2D), Tensoren (nD)
- Limitierung der Funktionalität steigert **Effizienz**
  - Speicherplatz komplett bei Anlegung reserviert
  - b Bytes pro Eintrag, konsekutiv (linear) im Arbeitsspeicher
  - Adresse jedes Eintrags dadurch einfach berechenbar

ndarrays

- Neues Konzept: **Array**, stark spezialisierte Variante von Listen
  - Alle Einträge eines Arrays vom gleichen Datentyp, bspw. `float`
  - Größe des Arrays generell variabel, bei der Erstellung bekannt und dann fest
  - Einträge einzeln modifizierbar
- In NumPy: `ndarray` für n-dimensionale Arrays
  - Vektoren (1D), Matrizen (2D), Tensoren (nD)
- Limitierung der Funktionalität steigert **Effizienz**
  - Speicherplatz komplett bei Anlegung reserviert
  - b Bytes pro Eintrag, konsekutiv (linear) im Arbeitsspeicher
  - Adresse jedes Eintrags dadurch einfach berechenbar
  - Sehr schneller Zugriff auf einzelne Elemente des Arrays

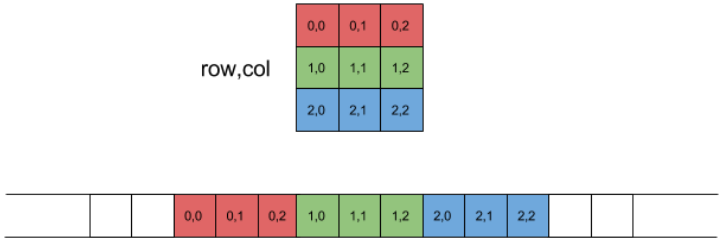
Einleitung



https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays, Eli Bendersky, unlicense.org

- NumPy kapselt all dies, insbesondere im Fall $n > 1$

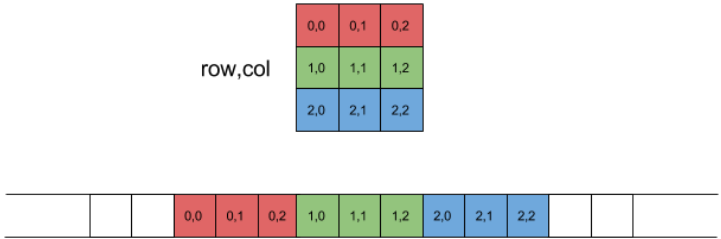
Einleitung



https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays, Eli Bendersky, unlicense.org

- NumPy kapselt all dies, insbesondere im Fall $n > 1$
- Ausreichend: Speicherung prinzipiell bekannt

Einleitung



https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays, Eli Bendersky, unlicense.org

- NumPy kapselt all dies, insbesondere im Fall $n > 1$
- Ausreichend: Speicherung prinzipiell bekannt
- Im Normalfall ignorierbares „technisches Detail“

Anlegen von Arrays in NumPy

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)
- Funktion `np.array()`

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)
- Funktion `np.array()`
- **Einfachste Möglichkeit**: Elemente des Vektors (der Matrix) als Liste (von Listen)

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)
- Funktion `np.array()`
- **Einfachste Möglichkeit**: Elemente des Vektors (der Matrix) als Liste (von Listen)
- Später im Abschnitt: bessere Möglichkeiten

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)
- Funktion `np.array()`
- **Einfachste Möglichkeit**: Elemente des Vektors (der Matrix) als Liste (von Listen)
- Später im Abschnitt: bessere Möglichkeiten
- Vektoren in NumPy: Zeilenvektoren, wenigstens beim Anlegen eines Arrays

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)
- Funktion `np.array()`
- **Einfachste Möglichkeit**: Elemente des Vektors (der Matrix) als Liste (von Listen)
- Später im Abschnitt: bessere Möglichkeiten
- Vektoren in NumPy: Zeilenvektoren, wenigstens beim Anlegen eines Arrays
- **Datentyp jedes Elements** aus den übergebenen Daten

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)
- Funktion `np.array()`
- **Einfachste Möglichkeit**: Elemente des Vektors (der Matrix) als Liste (von Listen)
- Später im Abschnitt: bessere Möglichkeiten
- Vektoren in NumPy: Zeilenvektoren, wenigstens beim Anlegen eines Arrays
- **Datentyp jedes Elements** aus den übergebenen Daten
  - Bei Mischungen automatisch allgemeinerer Datentyp

Anlegen von Arrays in NumPy

- `import numpy as np` (Benennung ist Konvention)
- Funktion `np.array()`
- **Einfachste Möglichkeit**: Elemente des Vektors (der Matrix) als Liste (von Listen)
- Später im Abschnitt: bessere Möglichkeiten
- Vektoren in NumPy: Zeilenvektoren, wenigstens beim Anlegen eines Arrays
- **Datentyp jedes Elements** aus den übergebenen Daten
  - Bei Mischungen automatisch allgemeinerer Datentyp
  - Alternativ: optionales Argument `dtype`

Codebeispiele

arange() und linspace()

arange() und linspace()

- Nützliche Funktionen für **spezielle Arrays für Iterationen**

arange() und linspace()

- Nützliche Funktionen für **spezielle Arrays für Iterationen**
- `arange(start, stop, step)`

arange() und linspace()

- Nützliche Funktionen für **spezielle Arrays für Iterationen**
- `arange(start, stop, step)`
  - NumPy-Pendant zur bekannten `range(start, stop, step)` Funktion

arange() und linspace()

- Nützliche Funktionen für **spezielle Arrays für Iterationen**
- `arange(start, stop, step)`
  - NumPy-Pendant zur bekannten `range(start, stop, step)` Funktion
  - Funktioniert exakt genauso, gibt `ndarray` zurück

arange() und linspace()

- Nützliche Funktionen für **spezielle Arrays für Iterationen**
- `arange(start, stop, step)`
  - NumPy-Pendant zur bekannten `range(start, stop, step)` Funktion
  - Funktioniert exakt genauso, gibt `ndarray` zurück
- `linspace(a, b, n)`

arange() und linspace()

- Nützliche Funktionen für **spezielle Arrays für Iterationen**
- `arange(start, stop, step)`
  - NumPy-Pendant zur bekannten `range(start, stop, step)` Funktion
  - Funktioniert exakt genauso, gibt `ndarray` zurück
- `linspace(a, b, n)`
  - Erstellt Unterteilung des Intervalls $[a, b]$ in n äquidistante Punkte

arange() und linspace()

- Nützliche Funktionen für **spezielle Arrays für Iterationen**
- `arange(start, stop, step)`
  - NumPy-Pendant zur bekannten `range(start, stop, step)` Funktion
  - Funktioniert exakt genauso, gibt `ndarray` zurück
- `linspace(a, b, n)`
  - Erstellt Unterteilung des Intervalls $[a, b]$ in n äquidistante Punkte
  - Intervallgrenzen **beide** inklusiv im Gegensatz zu `arange()`

Codebeispiele

Weitere hilfreiche Generator-Funktionen

Weitere hilfreiche Generator-Funktionen

- `np.zeros(n)`, `np.zeros( (n,m) )`: Vorbelegung mit Nullen

Weitere hilfreiche Generator-Funktionen

- `np.zeros(n)`, `np.zeros( (n,m) )`: Vorbelegung mit Nullen
- `np.ones(n)`, `np.ones( (n,m) )`: Vorbelegung mit Einsen

Weitere hilfreiche Generator-Funktionen

- `np.zeros(n)`, `np.zeros( (n,m) )`: Vorbelegung mit Nullen
- `np.ones(n)`, `np.ones( (n,m) )`: Vorbelegung mit Einsen
- `np.eye(n)`: $n \times n$ Einheitsmatrix

Weitere hilfreiche Generator-Funktionen

- `np.zeros(n)`, `np.zeros( (n,m) )`: Vorbelegung mit Nullen
- `np.ones(n)`, `np.ones( (n,m) )`: Vorbelegung mit Einsen
- `np.eye(n)`: $n \times n$ Einheitsmatrix
- `np.diag(...)`: Diagonal- und Bandmatrizen

Weitere hilfreiche Generator-Funktionen

- `np.zeros(n)`, `np.zeros( (n,m) )`: Vorbelegung mit Nullen
- `np.ones(n)`, `np.ones( (n,m) )`: Vorbelegung mit Einsen
- `np.eye(n)`: $n \times n$ Einheitsmatrix
- `np.diag(...)`: Diagonal- und Bandmatrizen
- `np.random.random(...)`: Zufallsmatrizen

Codebeispiele

Eigenschaften von ndarrays

Eigenschaften von ndarrays

- Größe und Form eines `ndarray`: Attribute `shape` und `size`

Eigenschaften von ndarrays

- Größe und Form eines `ndarray`: Attribute `shape` und `size`
- Analog (aber viel mächtiger/bequemer) zu `len()` für Listen

Eigenschaften von ndarrays

- Größe und Form eines `ndarray`: Attribute `shape` und `size`
- Analog (aber viel mächtiger/bequemer) zu `len()` für Listen
- Nach dem nächsten Codebeispiel: Attribut `shape` **beschreibbar** (!)

Codebeispiel

Array reshaping

Array reshaping

- Transposition von Matrizen

Array reshaping

- Transposition von Matrizen
- Änderung des `shape` Attributs: beinahe beliebige Änderung der Form eines `ndarray`

Array reshaping

- Transposition von Matrizen
- Änderung des `shape` Attributs: beinahe beliebige Änderung der Form eines `ndarray`
- Bspw. Vektor zu Matrix

Array reshaping

- Transposition von Matrizen
- Änderung des `shape` Attributs: beinahe beliebige Änderung der Form eines `ndarray`
- Bspw. Vektor zu Matrix
- Einschränkung: Anzahl der Elemente muss passen („glatt aufgehen muss“)

Array reshaping

- Transposition von Matrizen
- Änderung des `shape` Attributs: beinahe beliebige Änderung der Form eines `ndarray`
- Bspw. Vektor zu Matrix
- Einschränkung: Anzahl der Elemente muss passen („glatt aufgehen muss“)
- Alternative: `reshape()`

Array reshaping

- Transposition von Matrizen
- Änderung des `shape` Attributs: beinahe beliebige Änderung der Form eines `ndarray`
- Bspw. Vektor zu Matrix
- Einschränkung: Anzahl der Elemente muss passen („glatt aufgehen muss“)
- Alternative: `reshape()`
  - Oft bequemer

Array reshaping

- Transposition von Matrizen
- Änderung des `shape` Attributs: beinahe beliebige Änderung der Form eines `ndarray`
- Bspw. Vektor zu Matrix
- Einschränkung: Anzahl der Elemente muss passen („glatt aufgehen muss“)
- Alternative: `reshape()`
  - Oft bequemer
  - Aber ineffizienter: erzeugt ein neues Array

Array reshaping

- Transposition von Matrizen
- Änderung des `shape` Attributs: beinahe beliebige Änderung der Form eines `ndarray`
- Bspw. Vektor zu Matrix
- Einschränkung: Anzahl der Elemente muss passen („glatt aufgehen muss“)
- Alternative: `reshape()`
  - Oft bequemer
  - Aber ineffizienter: erzeugt ein neues Array
  - Verdopplung des Speicherverbrauchs

Codebeispiele

Impressum, Danksagung und Quellen

Stiftung Innovation in der Hochschullehre

Gefördert durch die Stiftung Innovation in der Hochschullehre im Rahmen des Projekts digit@L, https://stiftung-hochschullehre.de

SimTech

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie

---

Autor: Dominik Göddeke, IANS, Universität Stuttgart

CC BY

Weitere Quellen:

- Logos Universität Stuttgart, IANS, SimTech: Universität Stuttgart, alle Rechte vorbehalten
- Logo Python: https://freesvg.org/387, CC-0
- Logo Stiftung: Stiftung Innovation in der Hochschullehre, alle Rechte vorbehalten
- Logo ZOERR: Universität Tübingen, alle Rechte vorbehalten

ZOERR

Veröffentlicht auf dem Zentralen OER Repositorium Baden-Württemberg, https://www.zoerr.de