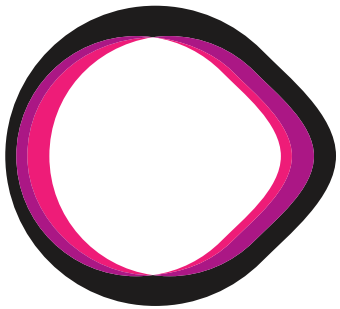
Python – Lektion 09
NumPy II



OST
Ostschweizer
Fachhochschule

Rückblick

- NumPy I

ndarray erzeugen

- ndarray erzeugen

```
>>> arr1 = np.array([1, 2, 3])
>>> print(arr1)
[1 2 3]

>>> arr2 = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(arr2)
[[1 2 3]
 [4 5 6]]

>>> arr2.ndim
2
>>> arr2.shape
(2, 3)
```

ndarray erzeugen

- Weitere Methoden, um Arrays zu erzeugen

Funktion	Resultat
`np.arange(0, 4, 0.5)`	`array([0, 0.5, 1, 1.5])`
`np.ones((2,2))`	`array([[1, 1], [1, 1]])`
`np.ones_like(arr1)`	`array([1, 1, 1])`
`np.zeros((2,2))`	`array([[0, 0], [0, 0]])`
`np.zeros_like(arr1)`	`array([0, 0, 0])`
`np.full((2,2), 7.0)`	`array([[7, 7], [7, 7]])`
`np.full_like(arr1, 7)`	`array([7, 7, 7])`
`np.eye(2)`	`array([[1, 0], [0, 1]])`
`np.identity(2)`	`array([[1, 0], [0, 1]])`
`np.linspace(0, 1, 5)`	`array([0, 0.25, 0.5, 0.75, 1])`
`np.logspace(0, 1, 4)`	`array([1, 2.1544, 4.6416, 10])`
`np.random.randn(3)`	`array([0.7576, 0.0135, -0.8934])`
`np.random.randint(0,10,3)`	`array([0, 5, 4])`

ndarray-Datentypen

- Datentyp wird automatisch ermittelt
  z.B. `np.int64` oder `np.float64`
- Datentyp erzwingen
  `np.array([1, 2, 3], dtype=np.complex)`
- Mögliche Datentypen

`np.int8, np.uint8`	8-Bit Ganzzahlen
`np.int16, np.uint16`	16-Bit Ganzzahlen
`np.int32, np.uint32`	32-Bit Ganzzahlen
`np.int64, np.uint64`	64-Bit Ganzzahlen
`np.float16`	Float mit halber Genauigkeit
`np.float32`	Float mit einfacher Genauigkeit
`np.float64`	Float mit doppelter Genauigkeit
`np.float128`	Float mit vierfacher Genauigkeit
`np.complex64/128/256`	Komplexe Zahl
`np.bool`	Boolescher Wert, True/False

Arithmetische Operationen

- Arithmetische Operationen werden elementweise ausgeführt

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

Operation	Resultat
`arr + arr`	`array([2., 4., 6.])`
`arr + 1`	`array([2., 3., 4.])`
`arr - arr`	`array([0., 0., 0.])`
`arr - 1`	`array([0., 1., 2.])`
`arr*arr`	`array([1., 4., 9.])`
`arr*2`	`array([2., 4., 6.])`
`arr/arr`	`array([1., 1., 1.])`
`arr/2`	`array([0.5., 1., 1.5])`
`arr**2`	`array([1., 4., 9.])`
`arr > 2`	`array([False, False, True], dtype=bool)`

Broadcasting

- Arithmetische Operationen können auch mit Arrays unterschiedlicher Shape vorgenommen werden.
- Dazu wird die Shape des kleineren Arrays automatisch auf die Shape des grösseren Arrays ausgebreitet (broadcasted).
- Das Broadcasting erfolgt nach bestimmten Regeln:
  - Falls ein Array weniger Dimensionen (ndim) hat als das andere, so werden dem Array mit weniger Dimensionen von links her so viele Dimensionen eingefügt, bis beide Arrays diesselbe Anzahl Dimensionen haben.
  - Falls die Shapes von zwei Arrays an einer Shape-Position nicht übereinstimmen, wird die Shape desjenigen Arrays angepasst, die eine 1 enthält. Der Wert wird dann auf den Wert des anderen Arrays erhöht.
  - Falls in irgendeiner Dimension die Grössen unterschiedlich sind und keine von beiden 1 ist, wird ein Fehler ausgegeben.

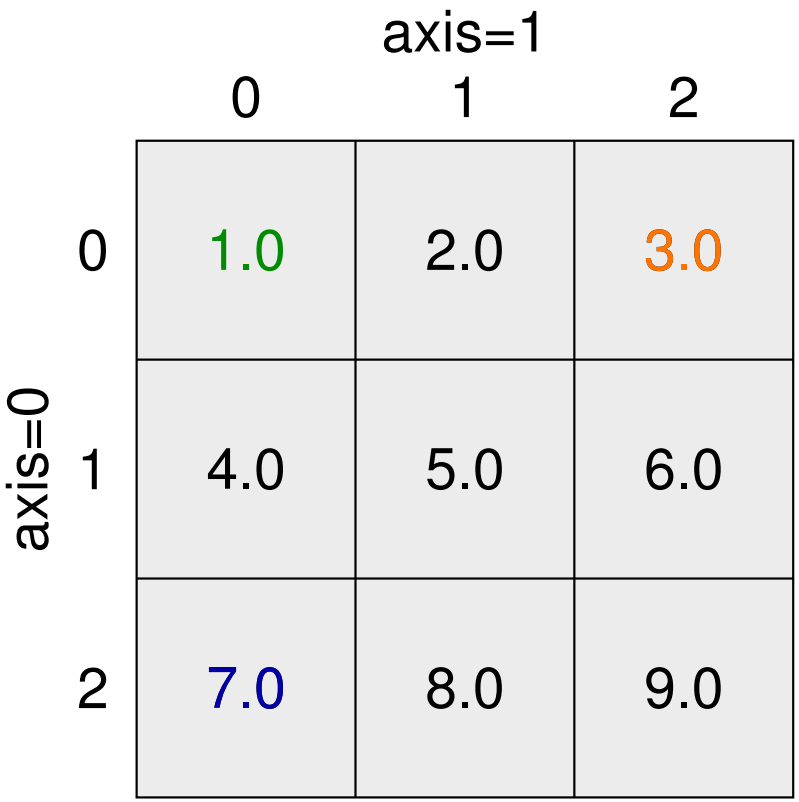
Indexierung

- Indexierung von 2D-Arrays

```
arr[axis0, axis1]
```

- Beispiele:

```
>>> arr[0, 0]
1.0
>>> arr[2, 0]
7.0
>>> arr[0, 2]
3.0
```



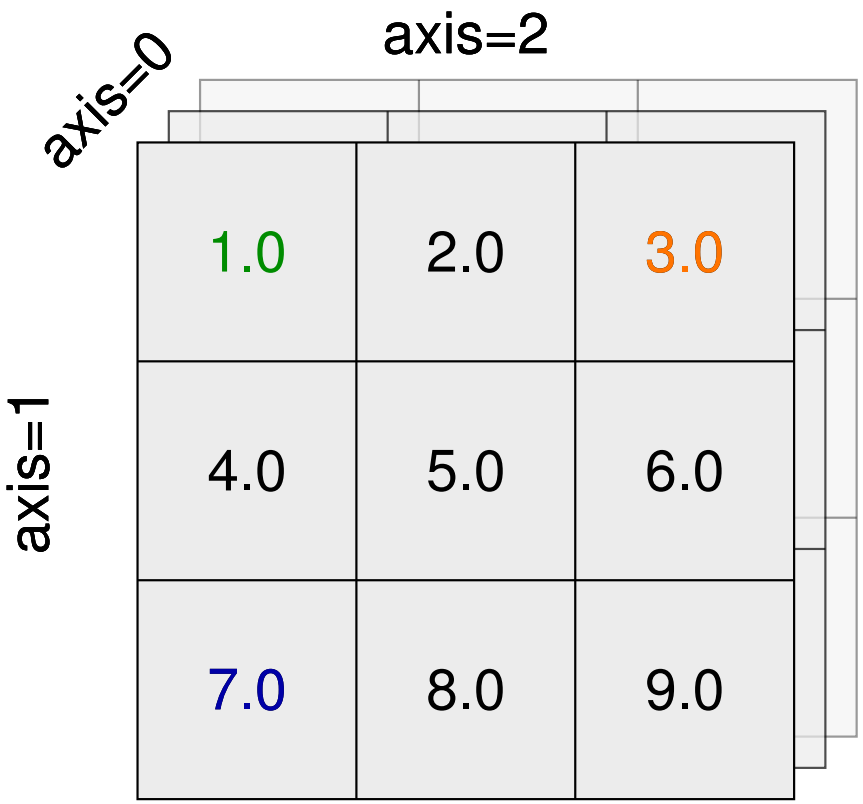
Indexierung

- Indexierung von 3D-Arrays

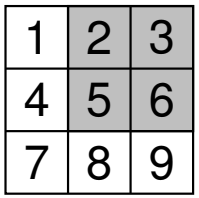
```
arr[axis0, axis1, axis2]
```

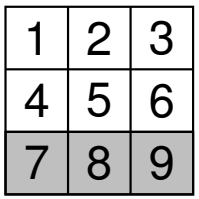
- Beispiele:

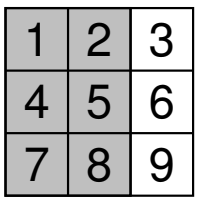
```
>>> arr[0, 0, 0]
1.0
>>> arr[0, 2, 0]
7.0
>>> arr[0, 0, 2]
3.0
```

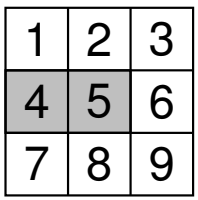


Slicing

`arr`	Ausdruck	Shape	Resultat
	`arr[:2, 1:]`	(2, 2)	`array([[2, 3], [5, 6]])`

	`arr[2]`
	`arr[2, :]`
	`arr[2:, :]`
		(3,)
		(3,)
		(1, 3)
			`array([7, 8, 9])`
			`array([7, 8, 9])`
			`array([[7, 8, 9]])`

	`arr[:, :2]`	(3, 2)	`array([[1, 2],`
			`[4, 5],`
			`[7, 8]])`

	`arr[1, :2]` `arr[1:2, :2]`	(2,) (1, 2)	`array([4, 5])` `array([[4, 5]])`

→ `ndim` bleibt erhalten, falls bei jeder `axis` ein ":" steht.

Slice == View

- Ein Slice ist immer eine Referenz, keine Kopie!

```
>>> arr = np.arange(8)
>>> arr
array([0, 1, 2, 3, 4, 5, 6, 7])

>>> s = arr[2:5]   # array([2, 3, 4])
>>> s[0] = 13      # modifiziert auch arr
>>> arr
array([0, 1, 13, 3, 4, 5, 6, 7])
```

- Kopien werden mit `.copy()` erzeugt:

```
>>> s = arr[2:5].copy()
```

Mathematische Funktionen

- NumPy beinhaltet viele mathematische Funktionen:
  `https://docs.scipy.org/doc/numpy/reference/routines.math.html`
  - `np.sin()`
  - `np.cos()`
  - `np.exp()`
  - `np.cumsum()`
  - `...`
- Diese Funktionen operieren über das gesamte Array

```
>>> t = np.linspace(1, 3, 5)
>>> np.exp(t)
array([2.718,  4.481,  7.389 , 12.182, 20.085])
>>> np.cumsum(t)
array([ 1. ,  2.5,  4.5,  7. , 10. ])
>>> np.mean(t)
2.0
```

Lineare Algebra

- Liste der Funktionen:

  `https://docs.scipy.org/doc/numpy/reference/routines.linalg.html`

- Matrizenmultiplikation (siehe Notebook für weitere Details)

```
>>> np.dot(M, v)
>>> M.dot(v)
>>> M @ v          # ab Python 3.5
```

- Matrix transponieren $\boldsymbol{M}^T$

```
>>> np.transpose(M)
>>> M.T
```

- Matrix invertieren $\boldsymbol{M}^{-1}$

```
>>> np.linalg.inv(M)
```

- Lineares Gleichungssystem $\boldsymbol{Ax} = \boldsymbol{b}$ lösen

```
>>> x = np.linalg.solve(A, b)
```

Heutige Themen

- NumPy II
  - Determinante, Rang, Spur, Eigenwerte
  - Produkte von Vektoren
  - Euklidische Norm
  - Manipulationsmethoden für Arrays

NumPy - Lineare Algebra II

- Determinante, Rang, Spur, Eigenwerte
- Produkte von Vektoren
- Euklidische Norm

`http://localhost:8888/notebooks/linalg2.ipynb`

NumPy - Manipulationsmethoden für Arrays

`http://localhost:`
`8888/notebooks/manipulationsmethoden.ipynb`

NumPy - Arrays erzeugen und zusammenfügen

- `np.concatenate(arrays, axis=0, out=None)`
  Arrays entlang einer bereits bestehenden Achse verbinden.
- `np.stack(arrays, axis=0, out=None)`
  Arrays entlang einer neuen Achse verbinden.
- `np.column_stack(tup)`
  1-D Arrays spaltenweise zusammenfügen.
- `np.row_stack(tup)`
  1-D Arrays zeilenweise zusammenfügen.
- `np.hstack(tup)`
  Arrays horizontal (spaltenweise) zusammenfügen.
- `np.vstack(tup)`
  Arrays vertikal (zeilenweise) zusammenfügen.
- `np.c_[]`
  Slice Objekte entlang der zweiten Achse zusammenfügen.
- `np.r_[]`
  Slice Objekte entlang der ersten Achse zusammenfügen.

- `np.reshape(shape, order='C')`

  Gibt ein Array zurück, das die gleichen Daten in einer neuen Form enthält.
- `np.resize(new_shape, refcheck=True)`

  Ändert die Form des Arrays in-place.
- `np.flatten(self, order='C')`

  Gibt eine flache Kopie der Matrix zurück.
- `np.ravel(self, order='C')`

  Gibt eine flache Matrix zurück.
- `np.tile(A, reps)`

  Konstruiert ein Array, in dem `A` so oft wiederholt wird, wie die Anzahl der Wiederholungen `reps` angibt.
- `np.repeat(a, repeats, axis=None)`

  Konstruiert ein Array, in dem die Elemente von `a` so oft wiederholt werden, wie die Anzahl der Wiederholungen `repeats` angibt.