

- Autor Pavel Tišnovský, Red Hat
- Email <ptisnovs 0x40 redhat 0x2e com>
- Datum 2019-10-05

Obsah přednášky (1)

- ▶ Programovací jazyk Python
- ▶ Knihovna NumPy
- ▶ Skalární datové typy
- ▶ Datová struktura ndarray
- ▶ Tvar (shape) n-dimenzionálního pole
- ▶ Konstruktory datové struktury ndarray
 - ◆ numpy.array
 - ◆ numpy.zeros
 - ◆ numpy.ones
 - ◆ numpy.eye
 - ◆ numpy.arange
 - ◆ numpy.linspace

Obsah přednášky (2)

- ▶ Základní operace s n-dimenzionálními poli
 - ◆ přetypování prvků v poli
 - ◆ zjištění tvaru pole
 - ◆ změna tvaru pole
 - ◆ výběr prvků v poli
 - ◆ slicing
- ▶ Další operace s n-dimenzionálními poli
 - ◆ operátory
 - ◆ násobení matic
 - ◆ determinant
 - ◆ inverzní matice
 - ◆ filtrace

Obsah přednášky (3)

- ▶ Využití matic
 - ◆ vyřešení systému lineárních rovnic
- ▶ Další podbalíčky, které nalezneme v knihovně ndarray
- ▶ Odkazy na další informační zdroje

Programovací jazyk Python

- ▶ Dnes jeden z nejpopulárnějších programovacích jazyků
 - ◆ viz například TIOBE programming community index
 - <https://www.tiobe.com/tiobe-index/>
 - ◆ Popř. statistika dostupná na OpenHubu
 - <https://www.openhub.net/languages/compare>
- ▶ Dostupnost na většině platforem
 - ◆ na některých MCU jako MicroPython
- ▶ P-y-t-h-o-n- –2– Python 3
 - ◆ všechny ukázky pro Python 3.6+

Programovací jazyk Python

- ▶ Typické použití Pythonu
 - ◆ Nástroje a utility ovládané z příkazového řádku
 - ◆ Aplikace s grafickým uživatelským rozhraním
 - ◆ Client-server
 - serverová část (Flask, Django, CherryPy, ...)
 - klient (Brython, spíše technologické demo)
 - ◆ Numerické výpočty, symbolické výpočty
 - NumPy
 - SciPy
 - Matplotlib
 - ◆ Moderní způsoby využití Pythonu
 - AI
 - Machine Learning (Deep Learning)
PyTorch
 - Big data
 - ◆ Tzv. „glue“ jazyk
 - ◆ Vestavětelný interpret Pythonu

Knihovna NumPy

- ▶ Výslovnosti
 - ◆ [nəmpɪ]
 - ◆ [nəmpɪ]
- ▶ Historie
 - ◆ matrix package
 - ◆ Numeric
 - ◆ NumPy
- ▶ Podpora pro n-dimenzionální pole
 - ◆ + nové funkce
 - ◆ + nové (přetížené) operátory
- ▶ Kooperace s dalšími knihovnami a frameworky
 - ◆ SciPy
 - ◆ Matplotlib
 - ◆ OpenCV

Skalární datové typy

<https://docs.scipy.org/doc/numpy/user/basics.types.html>

Rozsah	Formát	Popis	
	bool	uloženo po bajtech	True/ False
	int8 -128..127	celočíselný se znaménkem	
	int16	celočíselný se znaménkem	

-32768 .. 32767			
int32	celočíselný se znaménkem		
-2147483648 .. 2147483647			
int64	celočíselný se znaménkem		
-9223372036854775808 ..			
9223372036854775807			
uint8	celočíselný bez znaménka		
0 .. 255			
uint16	celočíselný bez znaménka		
0 .. 65535			
uint32	celočíselný bez znaménka		
0 .. 4294967295			
uint64	celočíselný bez znaménka		
0 .. 18446744073709551615			
float16 (half)	plovoucí řádová čárka		poloviční přesnost
float32 (single)	plovoucí řádová čárka		jednoduchá přesnost
float64 (double)	plovoucí řádová čárka		dvojitá přesnost
complex64 2×float32	komplexní číslo (dvojice)		
complex128 2×float64	komplexní číslo (dvojice)		

Kódy skalárních datových typů

- jednoznakové kódy je možné použít namísto jména typu

Formát	Kód
formát	kód
bool	'?'
int8	'b'
int16	'h'
int32	'i'
int64	'l'
uint8	'B'
uint16	'H'
uint32	'I'
uint64	'L'
float16	'e'
float32	'f'
float64	'd'
complex64	'F'

complex128	'D'
------------	-----

Datový typ single

Celkový počet bitů (bytů): 32 (4)
 Bitů pro znaménko: 1
 Bitů pro exponent: 8
 Bitů pro mantisu: 23

Datový typ double

Celkový počet bitů (bytů): 64 (8)
 Bitů pro znaménko: 1
 Bitů pro exponent: 11
 Bitů pro mantisu: 52

Datový typ float16

Celkový počet bitů (bytů): 16 (2)
 Bitů pro znaménko: 1
 Bitů pro exponent: 5
 Bitů pro mantisu: 10
 BIAS (offset exponentu): 15
 Přesnost: 5-6 číslic
 Maximální hodnota: 65504
 Minimální hodnota: -65504
 Nejmenší kladná nenulová hodnota: $5,960 \times 10^{-8}$
 Nejmenší kladná normalizovaná hodnota: $6,104 \times 10^{-5}$

Datová struktura ndarray

- ▶ Představuje obecné n-dimenzionální pole
- ▶ Interní způsob uložení dat zcela odlišný od Pythonovských seznamů či n-tic
 - ◆ „pohled“ na kontinuální blok hodnot
- ▶ Homogenní datová struktura
 - ◆ menší flexibilita
 - ◆ menší paměťové nároky
 - ◆ vyšší výpočetní rychlosť díky použití nativního kódu
 - ◆ obecně lepší využití cache a rychlejší přístup k prvkům

Datová struktura ndarray (pokračování)

- ▶ Základní strukturovaný datový typ knihovny NumPy
- ▶ Volitelný počet dimenzií
 - ◆ vektory
 - ◆ matice
 - ◆ pole s větším počtem dimenzií
- ▶ Volitelný typ prvků
- ▶ Volitelné uspořádání prvků
 - ◆ podle zvyklostí jazyka Fortran
 - ◆ podle zvyklostí jazyka C

Tvar (shape) n-dimenzionálního pole

- ▶ Popisuje organizaci a uspořádání prvků v poli
 - ◆ n-tice obsahující rozměry pole v jednotlivých dimenzích
- ▶ Příklady tvarů
 - ◆ (10,) - vektor s deseti prvky
 - ◆ (2, 3) - dvourozměrná matice se dvěma řádky a třemi sloupcí
 - ◆ (2, 3, 4) - trojrozměrné pole
- ▶ Tvar je možné zjistit
 - ◆ atribut „shape“
 - ◆ funkce numpy.shape
- ▶ Tvar je možné změnit
 - ◆ funkce numpy.reshape

Konstrukce n-dimenzionálních polí

- ▶ Několik typů konstruktorů
 - ◆ numpy.array
 - ◆ numpy.zeros
 - ◆ numpy.ones
 - ◆ numpy.eye
 - ◆ numpy.arange
 - ◆ numpy.linspace
- ▶ Konverzní funkce

Konstruktor numpy.array

- ▶ parametry

```
array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)
```

Order

Hodnota	Význam
'C'	prvky jsou interně uspořádány jako v jazyku C
'F'	prvky jsou interně uspořádány jako v jazyku Fortran
'A'	ponecháme na implementaci, který způsob uspořádání zvolit

Order - rozdíl v uspořádání

- ▶ 2D matice tak, jak ji vidí uživatel (logická struktura)

```
| 1 2 3 |
| 4 5 6 |
| 7 8 9 |
```

- ▶ Uložení v operační paměti

```
1 2 3 4 5 6 7 8 9 - 'C'
1 4 7 2 5 8 3 6 9 - 'F'
```

Příklady použití funkce numpy.array

```
# vytvoření pole ze seznamu
```

```
>>> numpy.array([1,2,3,4])
      ↓
array([1, 2, 3, 4])
```

Příklady použití funkce numpy.array

```
# vytvoření pole z generátoru 'range'
>>> numpy.array(range(10))
      ↓
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Příklady použití funkce numpy.array

```
# explicitní specifikace typu všech prvků pole
# (interně se provádí přetypování)
>>> numpy.array(range(10), dtype=numpy.float)
      ↓
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

Příklady použití funkce numpy.array

```
# explicitní specifikace uspořádání prvků pole
# (nemá velký význam pro 1D pole=vektory)
>>> numpy.array(range(10), order='C')
      ↓
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Příklady použití funkce numpy.array

```
# explicitní specifikace uspořádání prvků pole
# (nemá velký význam pro 1D pole=vektory)
>>> numpy.array(range(10), order='F')
      ↓
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Příklady použití funkce numpy.array

```
# vytvoření dvourozměrné matice
>>> numpy.array([[1,2,3],[4,5,6]])
      ↓
array([[1, 2, 3],
       [4, 5, 6]])
```

Konstruktor numpy.zeros

- ▶ Vektor nebo matice s nulovými prvky
- ▶ Poměrně častý požadavek v praxi
 - ◆ opět lze zvolit interní uspořádání prvků

Volání konstruktoru numpy.zeros

```
>>> zeros(shape, dtype=float, order='C')
```

Příklady použití konstruktoru numpy.zeros

```
# jednorozměrný vektor s jediným prvkem
>>> numpy.zeros(1)
      ↓
array([ 0.])
```

Příklady použití konstruktoru numpy.zeros

```
# jednorozměrný vektor s deseti prvky
>>> numpy.zeros(10)
      ↓
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

Příklady použití konstruktoru numpy.zeros

```
# matice o velikosti 5x5 prvků, každý prvek je typu float
>>> numpy.zeros((5,5))
      ↓
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

Příklady použití konstruktoru numpy.zeros

```
# matice o velikosti 5x5 prvků, každý prvek je typu int
>>> numpy.zeros((5,5), dtype=int)
      ↓
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

Příklady použití konstruktoru numpy.zeros

```
# použití komplexních čísel
>>> numpy.zeros((2,2), dtype=numpy.complex)
      ↓
array([[ 0.+0.j,  0.+0.j],
       [ 0.+0.j,  0.+0.j]])
```

Konstruktor numpy.ones

- ▶ Vektor či matice s prvky nastavenými na jedničku
- ▶ (nejedná se o jednotkovou matici!)
 - ◆ viz konstruktor numpy.eye

Volání konstruktoru numpy.ones

```
>>> numpy.ones(shape, dtype=None, order='C')
```

Příklady použití konstruktoru numpy.ones

```
# jednorozměrný vektor s deseti prvky
>>> numpy.ones(10)
      ↓
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

Příklady použití konstruktoru numpy.ones

```
# matice se třemi řádky a čtyřmi sloupci
>>> numpy.ones((3,4))
      ↓
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
```

Příklady použití konstruktoru numpy.ones

```
# matice se třemi řádky a čtyřmi sloupci
# s explicitní specifikací typu prvků
>>> numpy.ones((3,4), dtype=int)
      ↓
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

Příklady použití konstruktoru numpy.ones

```
# trojrozměrné pole s prvky typu int
>>> numpy.ones((3,4,5), dtype=int)
      ↓
array([[[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],
       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],
       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]]])
```

Příklady použití konstruktoru numpy.ones

```
# trojrozměrné pole s prvky typu int
# (oproti předchozímu příkladu se velikosti v jednotlivých dimenzích
# liší)
>>> numpy.ones((5,4,3), dtype=int)
      ↓
array([[[1, 1, 1],
        [1, 1, 1],
        [1, 1, 1],
        [1, 1, 1]],
```

```
[[1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1]],  
 [[1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1]],  
 [[1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1]],  
 [[1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1],  
 [1, 1, 1]]])
```

Příklady použití konstruktoru numpy.ones

```
# zde může být použití typu komplexní číslo možná poněkud překvapující,  
# ovšem stále platí, že 1=1+0j  
>>> numpy.ones((3,2),dtype=numpy.complex)  
      ↓  
array([[ 1.+0.j,  1.+0.j],  
       [ 1.+0.j,  1.+0.j],  
       [ 1.+0.j,  1.+0.j]])
```

Konstruktor numpy.eye

► Vytvoří se jednotková matice
► Uvádí se její velikost
► Lze ovšem vytvořit i nečtvercovou matici
>>> numpy.eye(1)
 ↓
array([[1.]])

Konstruktor numpy.eye

```
>>> numpy.eye(5)  
      ↓  
array([[1.,  0.,  0.,  0.,  0.],  
       [0.,  1.,  0.,  0.,  0.],  
       [0.,  0.,  1.,  0.,  0.],  
       [0.,  0.,  0.,  1.,  0.],  
       [0.,  0.,  0.,  0.,  1.]])  
  
>>> numpy.eye(2,10)  
      ↓  
array([[1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],  
       [0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

Funkce numpy.arange

► Array+range

- Podobné jako xrange/range
 - ◆ ovšem návratovou hodnotou je ndarray

Použití funkce numpy.arange

```
# při použití jednoho parametru má tento parametr význam hodnoty „stop“  
# vytvoří se vektor s prvky od 0 do „stop“ (kromě)  
>>> numpy.arange(10)  
      ↓  
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Použití funkce numpy.arange

```
# specifikace hodnot „start“ (včetně) a „stop“ (kromě)  
>>> numpy.arange(10, 20)  
      ↓  
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

Použití funkce numpy.arange

```
# třetí nepovinný parametr určuje krok použitý při generování prvků  
vektoru  
>>> numpy.arange(10, 20, 2)  
      ↓  
array([10, 12, 14, 16, 18])
```

```
# krok může být samozřejmě záporný  
>>> numpy.arange(20, 10, -2)  
      ↓  
array([20, 18, 16, 14, 12])
```

Použití funkce numpy.arange

```
# nemusíme zůstat pouze u celých čísel, protože pracovat je možné i s  
hodnotami  
# typu float a complex  
>>> numpy.arange(0.5, 0.1)  
      ↓  
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,  
       1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,  
       2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,  
       3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,  
       4.4,  4.5,  4.6,  4.7,  4.8,  4.9])
```

Použití funkce numpy.arange

```
# použití komplexních konstant  
>>> numpy.arange(0+0j, 10+10j, 2+0j)  
      ↓  
array([ 0.+0.j,  2.+0.j,  4.+0.j,  6.+0.j,  8.+0.j])
```

Funkce numpy.linspace

- Při znalosti první a poslední hodnoty ve vektoru

- Zadává se
 - ◆ počáteční hodnota
 - ◆ koncová hodnota
 - ◆ počet prvků vektoru (implicitně 50 prvků)

Použití funkce numpy.linspace

```
# pokud se nespecifikuje počet prvků, bude se předpokládat, že výsledný
# vektor má mít padesát prvků
>>> numpy.linspace(0, 1)
      ↓
array([ 0.          ,  0.02040816,  0.04081633,  0.06122449,  0.08163265,
       0.10204082,  0.12244898,  0.14285714,  0.16326531,  0.18367347,
       0.20408163,  0.2244898 ,  0.24489796,  0.26530612,  0.28571429,
       0.30612245,  0.32653061,  0.34693878,  0.36734694,  0.3877551 ,
       0.40816327,  0.42857143,  0.44897959,  0.46938776,  0.48979592,
       0.51020408,  0.53061224,  0.55102041,  0.57142857,  0.59183673,
       0.6122449 ,  0.63265306,  0.65306122,  0.67346939,  0.69387755,
       0.71428571,  0.73469388,  0.75510204,  0.7755102 ,  0.79591837,
       0.81632653,  0.83673469,  0.85714286,  0.87755102,  0.89795918,
       0.91836735,  0.93877551,  0.95918367,  0.97959184,  1.        ])
```

Použití funkce numpy.linspace

```
# zde explicitně specifikujeme, že výsledný vektor má mít deset prvků
# (tím, že se začíná od nuly, získáme krok 0.11111111...)
>>> numpy.linspace(0, 1, 10)
      ↓
array([ 0.          ,  0.11111111,  0.22222222,  0.33333333,  0.44444444,
       0.55555556,  0.66666667,  0.77777778,  0.88888889,  1.        ])
```

Použití funkce numpy.linspace

```
# zde explicitně specifikujeme, že výsledný vektor má mít jedenáct prvků
>>> numpy.linspace(0, 1, 11)
      ↓
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,
       1. ])
```

Použití funkce numpy.linspace

```
# sekvence hodnot samozřejmě může i klesat
>>> numpy.linspace(1, 0, 11)
      ↓
array([ 1. ,  0.9,  0.8,  0.7,  0.6,  0.5,  0.4,  0.3,  0.2,  0.1,
       0.        ])
```

Použití funkce numpy.linspace

```
# použít je možné i komplexní čísla
>>> numpy.linspace(0+0j, 1+0j, 10)
      ↓
array([ 0.00000000+0.j,  0.11111111+0.j,  0.22222222+0.j,
       0.33333333+0.j,
```

```
0.44444444+0.j, 0.55555556+0.j, 0.66666667+0.j,  
0.77777778+0.j,  
0.88888889+0.j, 1.00000000+0.j])
```

Použití funkce numpy.linspace

```
# použít je možné i komplexní čísla  
>>> numpy.linspace(0+0j, 0+1j, 10)  
↓  
array([ 0.+0.j , 0.+0.1111111j , 0.+0.2222222j ,  
0.+0.3333333j , 0.+0.44444444j , 0.+0.55555556j , 0.+0.66666667j ,  
0.+0.77777778j , 0.+0.88888889j , 0.+1.j ])
```

Použití funkce numpy.linspace

```
# další možnost použití komplexních čísel  
>>> numpy.linspace(0+0j, 1+1j, 10)  
↓  
array([ 0.00000000+0.j , 0.11111111+0.11111111j ,  
0.22222222+0.22222222j , 0.33333333+0.33333333j ,  
0.44444444+0.44444444j , 0.55555556+0.55555556j ,  
0.66666667+0.66666667j , 0.77777778+0.77777778j ,  
0.88888889+0.88888889j , 1.00000000+1.j ])
```

Přetypování prvků v poli

- ▶ Dva způsoby
 - ◆ konverzní funkce
 - numpy.float32()
 - numpy.int32()
 - numpy.complex128()
 - ...
 - ◆ použití metody astype

Přetypování prvků v poli

```
# přetypování  
>>> numpy.int64([1,2,3,4])  
↓  
array([1, 2, 3, 4], dtype=int32)  
  
# přetypování  
>>> numpy.float16([1,2,3,4])  
↓  
array([ 1., 2., 3., 4.], dtype=float16)
```

Přetypování prvků v poli

```
# vektor čísel typu float  
>>> numpy.linspace(0, 1, 10)  
↓  
array([ 0. , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
```

```
0.55555556, 0.66666667, 0.77777778, 0.88888889, 1. ])
```

```
# přetypování na vektor celých čísel  
>>> numpy.int32(numpy.linspace(0, 1, 10))  
↓  
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1], dtype=int32)
```

Použití metody astype

```
>>> a=numpy.arange(0, 10)  
>>> a.astype(numpy.complex64)  
↓  
array([ 0.+0.j, 1.+0.j, 2.+0.j, 3.+0.j, 4.+0.j, 5.+0.j, 6.+0.j,  
    7.+0.j, 8.+0.j, 9.+0.j], dtype=complex64)
```

Zjištění počtu dimenzí a tvaru pole

- ▶ Atribut ndim
- ▶ Atribut shape
- ▶ Funkce numpy.shape

Zjištění počtu dimenzí tvaru pole

```
# jednorozměrný vektor  
>>> a=numpy.array([1,2,3])
```

```
# počet dimenzí vektoru  
>>> a.ndim  
1
```

```
# tvar vektoru  
>>> a.shape  
(3,)
```

Zjištění základních informací o poli

```
# jednorozměrný vektor
```

```
>>> a=numpy.array([1,2,3])  
>>> a.dtype.name  
'int64'
```

```
>>> a.itemsize  
8
```

```
>>> a.size  
3
```

Tisk velkých polí

```
>>> print(numpy.arange(10000).reshape(100,100))  
↓  
[[ 0   1   2 ...  97  98  99]  
 [ 100 101 102 ... 197 198 199]  
 [ 200 201 202 ... 297 298 299]]
```

```
...  
[9700 9701 9702 ... 9797 9798 9799]  
[9800 9801 9802 ... 9897 9898 9899]  
[9900 9901 9902 ... 9997 9998 9999]]
```

Zjištění tvaru pole

```
# dvourozměrná matice  
>>> b=numpy.array([[1,2,3],[4,5,6]])  
  
# tvar matice  
>>> b.shape  
↓  
(2, 3)
```

Zjištění tvaru pole

```
# trojrozměrné pole  
>>> c=numpy.zeros((2,3,4))  
>>> c  
↓  
array([[[ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.]],  
      [[[ 0.,  0.,  0.,  0.],  
        [ 0.,  0.,  0.,  0.],  
        [ 0.,  0.,  0.,  0.],  
        [ 0.,  0.,  0.,  0.]]])  
      [ 0.,  0.,  0.,  0.]])# tvar trojrozměrného pole  
  
>>> c.shape  
↓  
(2, 3, 4)
```

Zjištění tvaru pole

```
# další trojrozměrné pole, tentokrát vytvořené explicitně  
>>> d=numpy.array([[[1,2], [3,4]], [[5,6], [7,8]]])  
>>> numpy.shape(d)  
↓  
(2, 2, 2)
```

Změna tvaru pole

- ▶ Funkce `numpy.reshape`
 - ◆ nevytváří nové pole s jiným tvarem
 - ◆ „pouze“ změna pohledu na pole
 - ◆ ⇒ nelze měnit počet prvků

Změna tvaru pole

```
# běžná matice se dvěma řádky a třemi sloupci  
>>> b=numpy.array([[1,2,3],[4,5,6]])
```

```
# změna tvaru matice na 3x2 prvky
>>> numpy.reshape(b,(3,2))
↓
array([[1, 2],
       [3, 4],
       [5, 6]])

# zde vlastně dostaneme původní matici
>>> numpy.reshape(b,(2,3))
↓
array([[1, 2, 3],
       [4, 5, 6]])
```

Změna tvaru pole

```
# vytvoření matice s jediným řádkem
>>> numpy.reshape(b,(1,6))
↓
array([[1, 2, 3, 4, 5, 6]])

# vytvoření matice s jediným sloupcem
>>> numpy.reshape(b,(6,1))
↓
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

Vliv parametru order na (zdánlivou) změnu tvaru pole

- ▶ Parametr „order“ použit u konstruktoru numpy.array
- ▶ Lze použít i u numpy.reshape
 - ◆ opět změna pohledu
 - ◆ nikoli reorganizace prvků v paměti

Vliv parametru order na (zdánlivou) změnu tvaru pole

```
# vyzkoušíme význam nepovinného parametru order
>>> numpy.reshape(m, (6,4))
↓
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])

>>> numpy.reshape(m, (6,4), order='C')
↓
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
```

```
[12, 13, 14, 15],  
[16, 17, 18, 19],  
[20, 21, 22, 23]])  
  
>>> numpy.reshape(m, (6,4), order='F')  
↓  
array([[ 0,  6, 12, 18],  
       [ 1,  7, 13, 19],  
       [ 2,  8, 14, 20],  
       [ 3,  9, 15, 21],  
       [ 4, 10, 16, 22],  
       [ 5, 11, 17, 23]])
```

Výběr prvků v poli

```
# jednorozměrná pole - vektory  
>>> a=numpy.arange(12)  
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])  
  
>>> a[0]  
0  
  
>>> a[5]  
5  
  
>>> a[-1]  
11  
  
>>> a[-5]  
7
```

Výběr prvků v poli

```
# dvourozměrná pole - matice  
>>> m=numpy.reshape(numpy.arange(12), (3,4))  
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11]])
```

```
>>> m[0]  
array([0, 1, 2, 3])
```

```
>>> m[0][2]  
2
```

```
>>> m[0,2]  
2
```

Výběr prvků pomocí indexů uložených v jiném poli

```
>>> a=numpy.arange(12)  
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])  
  
>>> b=numpy.array([1,2,9,8,5])
```

```
>>> a[b]
array([11, 12, 19, 18, 15])

>>> b=numpy.array([1,2,-1,8,5])
>>> a[b]
array([11, 12, 19, 18, 15])
```

Výběr prvků pomocí indexů uložených v jiném poli

```
# dtto ale s dvourozměrným polem
>>> m1=numpy.array([[1,2,3],[4,5,6],[7,8,9]])
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> m2=numpy.array([0,2,1])
>>> m1[m2]
array([[1, 2, 3],
       [7, 8, 9],
       [4, 5, 6]])
```

Slicing

```
>>> a=numpy.arange(12)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

>>> a[3:7]
array([3, 4, 5, 6])
```

Slicing - vynechání indexu/indexů

```
>>> a=numpy.arange(12)

>>> a[:7]
array([0, 1, 2, 3, 4, 5, 6])

>>> a[5:]
array([ 5,  6,  7,  8,  9, 10, 11])

>>> a[:]
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

Prázdný řez polem

```
>>> a=numpy.arange(12)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

>>> a[-4:-6]
array([], dtype=int64)
```

Řezy a záporné indexy

```
>>> a=numpy.arange(12)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
>>> a[-6:-4]
array([6, 7])

>>> a[-6:]
array([ 6,  7,  8,  9, 10, 11])

>>> a[: -4]
array([0, 1, 2, 3, 4, 5, 6, 7])
```

Řezy vícerozměrných polí

```
>>> m=numpy.reshape(numpy.arange(25), (5,5))
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])

>>> m[2:4,3]
array([13, 18])

>>> m[2:4,3:5]
array([[13, 14],
       [18, 19]])

>>> m[1:4,1:4]
array([[ 6,  7,  8],
       [11, 12, 13],
       [16, 17, 18]])
```



```
>>> m[-4:-2,-4:-2]
array([[ 6,  7],
       [11, 12]])
```

Specifikace kroku při provádění řezů - vektory

```
>>> a=numpy.arange(1,11)
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10])

>>> a[1:10:1]
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10])

>>> a[1:10:2]
array([ 2,  4,  6,  8, 10])

>>> a[1:10:3]
array([2, 5, 8])

>>> a[::3]
array([ 1,  4,  7, 10])
```

Specifikace kroku při provádění řezů - matice

```

>>> m1=numpy.reshape(numpy.arange(0,25), (5,5))
>>> m1
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])

>>> m1[0:5:2]
array([[ 0,  1,  2,  3,  4],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24]])

>>> m1[1::2]
array([[ 5,  6,  7,  8,  9],
       [15, 16, 17, 18, 19]])

```

Sudé sloupce, sudé řádky

```

>>> m1[::-2,::2]
array([[ 0,  2,  4],
       [10, 12, 14],
       [20, 22, 24]])

```

Operátory

- ▶ Základní operátory jsou přetížené
- ▶ Prvky matice + skalár

```

>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])
>>> a1+100
      ↓
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
>>> a1*2
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])

```

Operátory

```

>>> m=numpy.reshape(numpy.arange(25),(5,5))
>>> m%2
      ↓
array([[0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0]])

```

Operátory

- ▶ Prvky dvou matic

```

>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])

```

```
>>> a2=numpy.eye(3)
>>> a1+a2
    ↓
array([[ 2.,  2.,  3.],
       [ 4.,  6.,  6.],
       [ 7.,  8., 10.]])
► Různé kombinace
>>> a1 * 10 + a2 * 20
    ↓
array([[ 30.,  20.,  30.],
       [ 40.,  70.,  60.],
       [ 70.,  80., 110.]])
```

Modifikace matice s využitím operátorů += atd.

```
>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])
>>> a1
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> a1 += 100
>>> a1
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

Násobení matic

► Operátor @, nikoli *

```
>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])
>>> a2=numpy.eye(3)
```

► Násobení prvek po prvku

```
>>> a1*a2
    ↓
array([[1.,  0.,  0.],
       [0.,  5.,  0.],
       [0.,  0.,  9.]])
```

Násobení matic

► Maticový součin

```
>>> a1@a2
    ↓
array([[1.,  2.,  3.],
       [4.,  5.,  6.],
       [7.,  8.,  9.]])
```

► Změna prvku původně jednotkové matice

```
>>> a2[1][1]=-1
>>> a1@a2
    ↓
array([[ 1., -2.,  3.],
```

```
[ 4., -5.,  6.],  
 [ 7., -8.,  9.]])
```

Výpočet determinantu

```
>>> import numpy  
>>> import numpy.linalg  
  
>>> m=numpy.array([[0,1,0],[1,1,1],[0,1,1]])  
>>> m  
array([[0, 1, 0],  
       [1, 1, 1],  
       [0, 1, 1]])  
  
>>> numpy.linalg.det(m)  
-1.0
```

Výpočet inverzní matice

```
>>> m=numpy.array([[0,1,0],[1,1,1],[0,1,1]])  
  
>>> numpy.linalg.inv(m)  
array([[ 0.,  1., -1.],  
       [ 1.,  0.,  0.],  
       [-1.,  0.,  1.]])
```

Testování - výsledkem musí být jednotková matice

```
>>> m=numpy.array([[0,1,0],[1,1,1],[0,1,1]])  
>>> m2=numpy.linalg.inv(m)  
  
>>> numpy.dot(m,m2)  
array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.]])
```

Singulární matice

```
>>> m5 = numpy.array([[100, 101, 102, 103, 104],  
                   [105, 106, 107, 108, 109],  
                   [110, 111, 112, 113, 114],  
                   [115, 116, 117, 118, 119],  
                   [120, 121, 122, 123, 124]])  
  
>>> m5inv=numpy.linalg.inv(m5)  
LinAlgError      Traceback (most recent call last)  
...  
...  
...  
LinAlgError: Singular matrix
```

Relační operátory: pole vs skalárni hodnota

```
>>> a=numpy.arange(12)
```

```
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

>>> a==5
array([False, False, False, False,  True, False, False, False,
       False], dtype=bool)

>>> a<6
array([ True,  True,  True,  True,  True,  True, False, False,
       False, False], dtype=bool)
```

Relační operátory: pole vs pole

```
>>> a=numpy.arange(1,11)
>>> b=numpy.array([100,0,100,0,100,0,100,0,100,0])

>>> a
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])

>>> b
array([100,    0, 100,    0, 100,    0, 100,    0, 100,    0])

>>> a==b
array([False, False, False, False, False, False, False, False,
       False], dtype=bool)

>>> a!=b
array([ True,  True,  True,  True,  True,  True,  True,  True,
       True], dtype=bool)

>>> a<b
array([ True, False,  True, False,  True, False,  True, False,
       True], dtype=bool)
```

Dílko pro matice

```
>>> m=numpy.arange(24)
>>> m
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
       16,
       17, 18, 19, 20, 21, 22, 23])

>>> x=numpy.reshape(m, (6,4), order='F')

>>> x<10
array([[False, False,  True,  True],
       [False,  True,  True,  True]], dtype=bool)

>>> x%2==1
```

```
array([[False, False, False, False],
       [ True,  True,  True,  True],
       [False, False, False, False],
       [ True,  True,  True,  True],
       [False, False, False, False],
       [ True,  True,  True,  True]], dtype=bool)
```

Výběr prvků na základě podmínky - filtrace

```
>>> a=numpy.arange(12)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

>>> a<6
array([ True,  True,  True,  True,  True,  True, False, False, False,
       False, False, False], dtype=bool)

>>> a[a<6]
array([0, 1, 2, 3, 4, 5])

>>> a[a%2 == 0]
array([ 0,  2,  4,  6,  8, 10])
```

Ovšem pozor u vícerozměrných polí

```
>>> m1=numpy.reshape(numpy.arange(100,125),(5,5))
>>> m1
array([[100, 101, 102, 103, 104],
       [105, 106, 107, 108, 109],
       [110, 111, 112, 113, 114],
       [115, 116, 117, 118, 119],
       [120, 121, 122, 123, 124]])
```

► Filtrací zíkáme jednorozměrný vektor

```
m1[m1%2 == 0]
array([100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124])
```

Další užitečné funkce - min, max, sum

```
>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])
      ↓
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> a1.max()
9

>>> a1.min()
1

>>> a1.sum()
45
```

Výběr osy ve funkčích min, max a sum

```
>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])
      ↓
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

► Sloupce

```
>>> a1.max(axis=0)
array([7, 8, 9])
```

```
>>> a1.sum(axis=0)
array([12, 15, 18])
```

► Řádky

```
>>> a1.max(axis=1)
array([3, 6, 9])
```

```
>>> a1.sum(axis=1)
array([ 6, 15, 24])
```

Univerzální funkce a operátory

```
>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])
>>> numpy.abs(a1-5)
      ↓
array([[4, 3, 2],
       [1, 0, 1],
       [2, 3, 4]])
```

Univerzální funkce a operátory

► Padesát hodnot v zadaném intervalu

```
>>> a=numpy.linspace(0, numpy.pi/2)
      ↓
array([ 0.          ,  0.03205707,  0.06411414,  0.0961712 ,  0.12822827,
       0.16028534,  0.19234241,  0.22439948,  0.25645654,  0.28851361,
       ...
       1.44256806,  1.47462512,  1.50668219,  1.53873926,  1.57079633])
```

► Výpočet sinů těchto hodnot

```
>>> numpy.sin(a)
      ↓
array([ 0.          ,  0.03205158,  0.06407022,  0.09602303,  0.12787716,
       0.1595999 ,  0.19115863,  0.22252093,  0.25365458,  0.28452759,
       ...
       0.99179001,  0.99537911,  0.99794539,  0.99948622,  1.          ])
```

Funkce vyššího řádu apply_along_axis

► (Anonymní) funkce, které se předává řádek/sloupec/matice n-1 dimenze

► Opět se specifikuje osa

```
>>> a1=numpy.array([[1,2,3], [4,5,6], [7,8,9]])
```

```
>>> numpy.apply_along_axis(lambda v:v[1], 0, a1)
array([4, 5, 6])
```

```
>>> numpy.apply_along_axis(lambda v:v[1], 1, a1)
array([2, 5, 8])
```

Vyřešení systému lineárních rovnic

- ▶ Triviální příklad - jedna rovnice o jedné neznámé
- ▶ Rovnice $2x = 10$
- ▶ Maticově
 - ◆ levá strana rovnice
 - ◆ pravá strana rovnice

```
▶ Řešení lze získat následovně
# levá strana rovnice (koeficienty)
>>> a=numpy.array([[2]])
```

```
# pravá strana rovnice
>>> b=numpy.array([10])
>>> numpy.linalg.solve(a,b)
↓
array([ 5.])
```

Vyřešení systému lineárních rovnic

- ▶ Dvě rovnice o dvou neznámých
 - $x + y = 2$
 - $x - y = 0$
 - ▶ Maticově
 - ◆ levé strany rovnic
 - ◆ pravé strany rovnic
- ```
▶ Řešení lze získat následovně
matice koeficientů původních rovnic
[1,1] znamená $1*x + 1*y$
>>> a=numpy.array([[1,1] , [1,-1]])
```

```
matice pravých stran rovnic
>>> b=numpy.array([2,0])
```

```
výpočet
>>> numpy.linalg.solve(a,b)
↓
array([1., 1.])
↓
x=1 a y=1
```

## Poněkud složitější příklad

---

- ▶ Zadání
  - $2x_1 + 3x_2 + 7x_3 = 47$
  - $3x_1 + 8x_2 + x_3 = 50$
  - $3x_2 + 3x_3 = 27$

► Řešení

```
>>> a=numpy.array([[2,3,7],[3,8,1],[0,3,3]])
>>> b=numpy.array([47,50,27])
>>> numpy.linalg.solve(a,b)
 ↓
array([2., 5., 4.])
```

Další podbalíčky, které nalezneme v knihovně ndarray

| Podbalíček podbalíčku | Stručný popis                                                         |
|-----------------------|-----------------------------------------------------------------------|
| doc                   | obsahuje dokumentaci ke knihovně i k základním konstrukcím a operacím |
| lib                   | základní funkce používané i některými dalšími podbalíčky              |
| random                | funkce pro využití generátorů pseudonáhodných číselných hodnot        |
| linalg                | funkce z oblasti lineární algebry                                     |
| fft                   | rychlá Fourierova transformace a pomocné funkce                       |
| polynomial            | funkce pro práci s polynomy                                           |
| testing               | nástroje pro psaní testů                                              |
| f2py                  | (jednosměrné) rozhraní mezi Fortranem a Pythonem                      |
| distutils             | další pomocné nástroje, které přímo nesouvisí s výpočty               |
| balíčkování           | nad vektory a maticemi, ale se způsobem                               |

Odkazy na další informační zdroje

- \* NumPy Home Page  
<http://www.numpy.org/>
- \* NumPy v1.10 Manual  
<http://docs.scipy.org/doc/numpy/index.html>
- \* NumPy (Wikipedia)  
<https://en.wikipedia.org/wiki/NumPy>