Języki i techniki programowania

Wykład 5

Maciej Rybczyński

Dlaczego NumPy jest użyteczny?

```
import numpy as np

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

print(a*10)
print(a/2)
print(a**2)
print(a * a)
print(a + 10.5)
print(np.cos(a))   # np.cos(a) - używa funkcji z biblioteki NumPy
print(a > 0)
```

operatory arytmetyczne zastosowane „na tablicach”
działają **w odniesieniu do elementów**!

```
[10 20 30]
[0.5 1.  1.5]
[1 4 9]
[1 4 9]
[11.5 12.5 13.5]
[ 0.54030231 -0.41614684 -0.9899925 ]
[ True  True  True]
```

Dlaczego NumPy jest użyteczny?

```
import numpy as np

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

b = np.array([5.0, 4.0, 3.0])

print(a + b)
print(a * b)
print(b / a)
print(a > b)
print((a - b) == 0)
```

```
[6. 6. 6.]
[5. 8. 9.]
[5. 2. 1.]
[False False False]
[False False  True]
```

Liczby losowe z zadanych rozkładów

lec_5a.py - E:/CERNBox/zajecia/Inzynieria_danych/Python/moje/wykla

File Edit Format Run Options Window Help

```
import numpy as np

print(np.random.normal(0,5, (2,3)))
print()

print(np.random.exponential(0.5, (2,4)))
print()

print(np.random.poisson(5, 10))
```

μ, σ

β

λ

$$\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$\exp(-x/\beta),\ x > 0$$

$$\frac{\lambda^n}{n!}e^{-\lambda}$$

```
= RESTART: E:/CERNBox/zajecia/Inzynieria_danych/Pythor
[[-2.81730147 -0.22282462 -3.43308929]
 [ 7.92863133 -0.04377347  1.01045618]]

[[1.75210091 0.04863863 0.38631474 1.80913141]
 [0.38397384 0.87583817 0.08167833 0.01244621]]

[7 4 5 4 7 6 5 3 6 2]
```

Liczby losowe – permutation, choice

lec_5b.py - E:/CERNBox/zajecia/Inzynieria_danych/Python/moje/wyklad_5/lec_5

File Edit Format Run Options Window Help

```
import numpy as np

print(np.random.permutation([1,10,100,1000]))
print(np.random.permutation(np.arange(1,11)))
print(np.random.permutation(['a','b','c']))
print('\n')

print(np.random.choice(['a','b','c','d']))
print(np.random.choice(['a','b','c','d'],2))
print(np.random.choice(['a','b','c','d'],(2,10)))
```

```
= RESTART: E:/CERNBox/zajecia/Inzynieria_danych/P
[   1 1000   10  100]
[ 5  2  4  8  7  1  3 10  9  6]
['b' 'c' 'a']


c
['b' 'c']
[['d' 'b' 'a' 'a' 'b' 'a' 'd' 'c' 'b' 'b']
 ['a' 'd' 'd' 'd' 'd' 'b' 'b' 'd' 'd' 'a']]
```

Tablice ze zmiennymi różnych typów

lec_5c.py - E:/CERNBox/zajecia/Inzynieria_danych/Python/moje/w

File Edit Format Run Options Window Help

```
import numpy as np

a = np.array([1,2,3])
print(a.dtype.name)

b = np.array([1.0,2.0,3.0])
print(b.dtype.name)

c = a + b
print(c)
print(c.dtype.name)
```

```
= RESTART: E:/CERNBox/zajecia
int32
float64
[2. 4. 6.]
float64
```

Typ wynikowej tablicy odpowiada bardziej ogólnemu lub precyzyjnemu (upcasting).

Nie działa z +=, *= itp.

sum(axis=...)

lec_5d.py - E:/CERNBox/zajecia/Inzynieria_danych/Python/moje/

File Edit Format Run Options Window Help

```
import numpy as np

a = np.arange(0,12).reshape(3,4)

print(a,'\n')

print(a.sum())

print(a.sum(axis=0))

print(a.sum(axis=1))
```

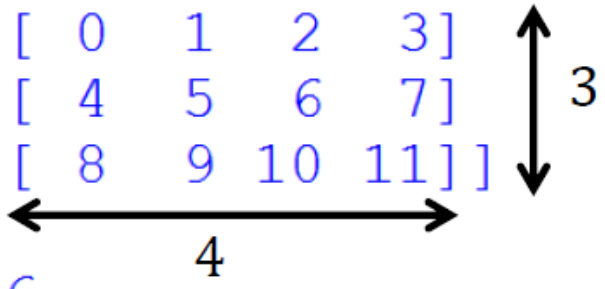
3 wiersze, 4 kolumny

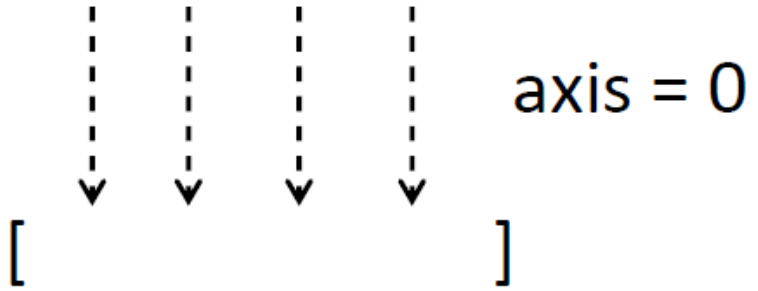
suma wszystkich elementów

sumowanie każdej z kolumn

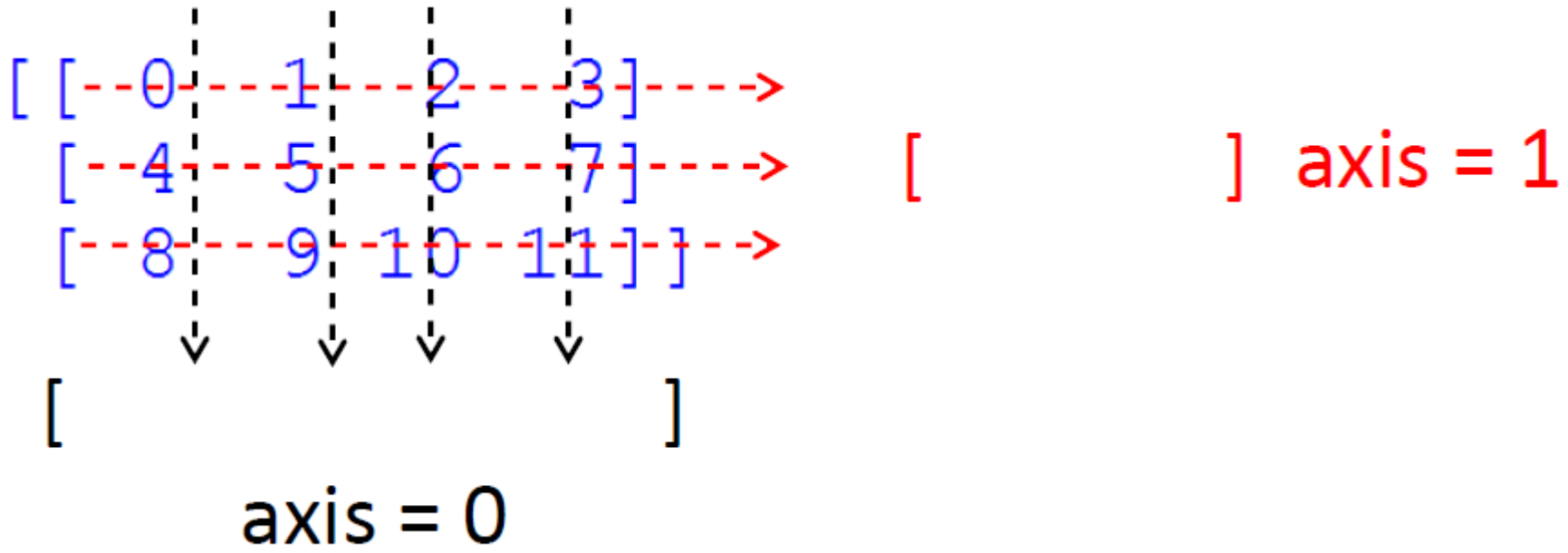
sumowanie każdego z wierszy

```
>>>
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
66
[12 15 18 21]
[ 6 22 38]
```





axis=0, axis=1



min(axis=..), max(axis=...)

lec_5e.py - E:/CERNBox/zajecia/Inzynieria_danych/Python

File Edit Format Run Options Window Help

```
import numpy as np

a = np.arange(0,12).reshape(3,4)

print(a,'\n')

print(a.min())

print(a.min(axis=0))

print(a.min(axis=1))
```

```
= RESTART: E:/CERNBox/zaje
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

0
[0 1 2 3]
[0 4 8]
```

min z wszystkich elementów
min z elementów każdej kolumny
min z elementów każdego wiersza

Inny sposób:
np.min(a)
np.min(a, axis=0)
np.min(a, axis=1)

np.sort(axis=...)

```
import numpy as np

a = np.array([[8,4,1],[2,1,0],[3,13,2]])
print(a,'\n')
print(np.sort(a,axis=0))
print('\n')
print(np.sort(a,axis=1))
```

sortowanie elementów każdej kolumny

sortowanie elementów każdego wiersza

```
= RESTART: E:/CERNBox/
[[ 8  4  1]
 [ 2  1  0]
 [ 3 13  2]]

[[ 2  1  0]
 [ 3  4  1]
 [ 8 13  2]]



[[ 1  4  8]
 [ 0  1  2]
 [ 2  3 13]]
```

identity()

```
import numpy as np

a = np.identity(4)
print(a,'\n')

a = np.identity(7, dtype=int)
print(a)
```

```
= RESTART: E:/CERNBox/z
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

[[1 0 0 0 0 0 0]
 [0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0]
 [0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0]
 [0 0 0 0 0 0 1]]
```

zbyt duża tablica

```
import numpy as np

# np.set_printoptions(threshold=50)

a = np.arange(10000)
print(a.reshape(100,100))
```

```
= RESTART: E:/CERNBox/zajecia/Inzynieria_d
[[   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]]
```

Jeśli tablica jest zbyt duża, NumPy pomija część środkową i drukuje tylko rogi. Aby wyłączyć, użyj set_printoptions.

linspace()

```
import numpy as np

x = np.linspace(0, 1, 6)
print(x)
```

6 liczb od 0 do 1

```
= RESTART: E:/CERNBox/zajeci
[0.  0.2 0.4 0.6 0.8 1. ]
```

iteracje

```
import numpy as np

a = np.arange(20)**2
a = a.reshape(5,4)
print(a,'\n')

for i in a:
    print(i)
```

iteracje po wierszach

```
= RESTART: E:/CERNBox/z
[[  0   1   4   9]
 [ 16  25  36  49]
 [ 64  81 100 121]
 [144 169 196 225]
 [256 289 324 361]]

[0 1 4 9]
[16 25 36 49]
[ 64  81 100 121]
[144 169 196 225]
[256 289 324 361]
```

Iteracje, flat

```
import numpy as np

a = np.arange(20)**2
a = a.reshape(5,4)
print(a,'\n')

for i in a.flat:
    print(i, end=' ')
```

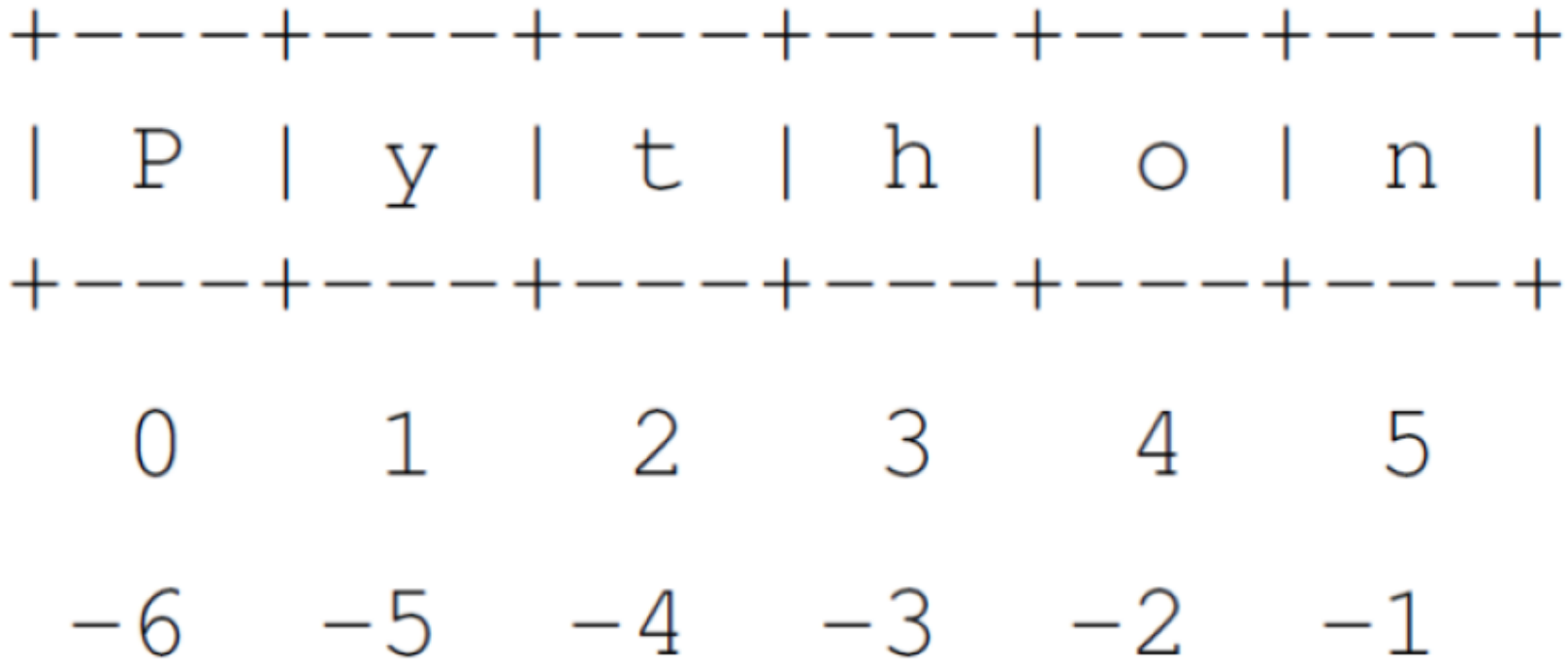
iteracje po wszystkich elementach

ten sam efekt:

```
for i in a:
    for el in i:
        print(el, end=' ')
```

```
= RESTART: E:/CERNBox/zajecia/Inzynieria_danych/Python/moje/wyklad
[[  0   1   4   9]
 [ 16  25  36  49]
 [ 64  81 100 121]
 [144 169 196 225]
 [256 289 324 361]]

0 1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361
```



s[2:5] = ’tho’

włączony

wyłączony

Slicing (krajanie na plastry)

```
import numpy as np

a = np.arange(5)**2
print(a,'\n')

print(a[1], a[-2])

print(a[2:4])

print(a[:3])

print(a[3:])

print(a[:])
```

```
= RESTART: E:/CEI
[ 0  1  4  9 16]

1 9
[4 9]
[0 1 4]
[ 9 16]
[ 0  1  4  9 16]
```

a[: 3] = a[0 : 3]

a[3 :] = a[3 : len(a)] (do ostatniego elementu)

a[:] = a[0 : len(a)]

Slicing (krajanie na plastry)

```
import numpy as np

a = np.arange(7)**2
print(a,'\n')

print(a[2:6:2])

print(a[ : :3])

print(a[ : :-1])

a[2:6]=0

print(a)
```

co drugi element

a[::3]=a[0:len(a):3] (co trzeci element)

od końca

```
= RESTART: E:/CERNBox/zaje
[ 0  1  4  9 16 25 36]

[ 4 16]
[ 0  9 36]
[36 25 16  9  4  1  0]
[ 0  1  0  0  0  0 36]
```

Slicing (krajanie na plastry)

```
import numpy as np

a = np.arange(20)**2
a = a.reshape(5,4)

print(a,'\n')
```

```
[[  0   1   4   9]
 [ 16  25  36  49]
 [ 64  81 100 121]
 [144 169 196 225]
 [256 289 324 361]]

121
```

drugi wiersz, trzecia kolumna

```
print(a[2,3],'\n')
```

wiersze, kolumny

```
print(a[0:3,0:2],'\n')
```

```
[[ 0  1]
 [16 25]
 [64 81]]
```

wiersze, wszystkie kolumny

```
print(a[0:3, :])
```

```
[[  0   1   4   9]
 [ 16  25  36  49]
 [ 64  81 100 121]]
```

kopiowanie

```
import numpy as np

a = np.random.randint(0,10,6)
print(a,'\n')

b = a

b[0] = 0

print(a)
```

```
= RESTART: E:/CERN
[7 0 0 3 2 4]


[0 0 0 3 2 4]
```

żaden nowy obiekt nie jest tworzony,
a i b odnoszą się do tego samego obiektu.
To samo z b = a[2 : 4] (widok)

kopiowanie

```
import numpy as np

a = np.random.randint(0,10,6)
print(a,'\n')

b = a.copy()

b[:] = 0

print(a,b)
```

tworzony jest nowy obiekt

```
[0 5 6 9 8 4]

[0 5 6 9 8 4] [0 0 0 0 0 0]
```

newaxis

```
import numpy as np

a = np.random.randint(0,10,6)
print(a,'\n')

b = a[ : ,np.newaxis]

print(b)
```

tworzony jest wektor

```
[3 9 8 4 1 2]

[[3]
 [9]
 [8]
 [4]
 [1]
 [2]]
```

sztuczka logiczna

```
import numpy as np

a = np.random.randint(0,10,(2,8))
print(a,'\n')

b = (a >=5)
print(b,'\n')

a[b] = 0
print(a)
```

```
[[7 5 8 9 3 1 7 2]
 [9 4 8 0 8 8 8 5]]

[[ True  True  True  True False False  True False]
 [ True False  True False  True  True  True  True]]

[[0 0 0 0 3 1 0 2]
 [0 4 0 0 0 0 0 0]]
```

zaokrąglanie (round)

```
import numpy as np

a = np.random.uniform(0,10,(2,4))
print(a,'\n')

print(np.round(a, 3))
```

```
[[9.21455649 6.23858274 8.49245012 6.18910808]
 [0.34267501 8.43116017 9.98883576 1.13208015]]

[[9.215 6.239 8.492 6.189]
 [0.343 8.431 9.989 1.132]]
```

Zobacz:
https://numpy.org/doc/stable/reference/routines.html
funkcje NumPy według kategorii