

# Programowanie obiektowe

## W110PA-SI0054L (INP001045L)

### rok akademicki 2023/24

### semestr zimowy

## Laboratorium 6

Karol Tarnowski

[karol.tarnowski@pwr.edu.pl](mailto:karol.tarnowski@pwr.edu.pl)

L-1 p. 220

# Plan

## Biblioteka `numpy`

## Klasa `ndarray`

- tworzenie tablic
- właściwości tablic
- typy danych tablic
- dostęp do elementów tablic
- obliczenia na tablicach

# Biblioteka numpy

- NumPy jest biblioteką pythona, która umożliwia pracę z wielowymiarowymi tablicami danych
- Jest to biblioteka kluczowa dla wielu programów zaimplementowanych w pythonie służących do prowadzenia obliczeń naukowych i/lub analizy danych

# Klasa `ndarray`

- Kluczową klasą zaimplementowaną w bibliotece **NumPy** jest klasa `ndarray` reprezentująca wielowymiarową tablicę danych
- Na obiektach `ndarray` o zgodnych rozmiarach można wykonywać działania element po elemencie

# Klasa ndarray

Przykład pokazuje:

- Importowanie biblioteki **numpy**
- tworzenie obiektów **ndarray** z list
- proste działania wykonywane na tablicach

```
[1, 2, 3, 2, 3, 4]  
<class 'numpy.ndarray'>  
[3 5 7]
```

```
numpy_01.py X  
1  # -*- coding: utf-8 -*-  
2  """  
3  Prosty przykład użycia biblioteki numpy  
4  """  
5  
6  # importowanie biblioteki numpy  
7  import numpy as np  
8  
9  # "dodawanie" list jest ich konkatenacją  
10 a = [1, 2, 3]  
11 b = [2, 3, 4]  
12 print(a + b)  
13  
14 # funkcja array tworzy tablicę danych  
15 npa = np.array(a)  
16 npb = np.array(b)  
17  
18 # biblioteka numpy zawiera definicję klasy ndarray  
19 # reprezentującej tablicę danych  
20 print(type(npa))  
21  
22 # tablice ndarray można np. dodawać element po elemencie  
23 npc = npa + npb  
24 print(npc)
```

# Klasa `ndarray`

Atrybutami klasy `ndarray` są (m. in.)

- **`shape`** - krotka zawierająca informacje o rozmiarach tablicy
- **`size`** - liczba elementów tablicy
- **`ndim`** - liczba wymiarów tablicy

Więcej informacji o klasie `ndarray`:

<https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html>

# Klasa ndarray

```
numpy_02.py X
6  import numpy as np
7
8  # utworzenie wektora o trzech elementach
9  a = np.array([1,2,3])
10 print("type(a)", type(a))
11 print("a =", a)
12
13 # pole shape - krotka zawierajaca rozmiar tablicy
14 print("a.shape =", a.shape)
15
16 b = np.array([[1, 2, 3], [4, 5, 6]])
17 print("b =", b)
18 print("b.shape =", b.shape)
19
20 # pole size - liczba elementow tablicy
21 print("a.size =", a.size)
22 print("b.size =", b.size)
23
24 # pole ndim - liczba wymiarow
25 print("a.ndim =", a.ndim)
26 print("b.ndim =", b.ndim)
27
```

```
type(a) <class 'numpy.ndarray'>
a = [1 2 3]
a.shape = (3,)
b = [[1 2 3]
      [4 5 6]]
b.shape = (2, 3)
a.size = 3
b.size = 6
a.ndim = 1
b.ndim = 2
```

# Klasa ndarray

```
numpy_02.py X
28 # pojedyncza liczba jest zamieniana
29 # w tablicę zerowymiarową
30 c = np.array(7)
31 print("c =", c)
32 print("c.shape =", c.shape)
33 print("c.size =", c.size)
34 print("c.ndim =", c.ndim)
35
```

```
c = 7
c.shape = ()
c.size = 1
c.ndim = 0
```



# Klasa ndarray

Przykładowe sposoby tworzenia tablic

```
numpy_03.py X
7  import numpy as np
8
9  # tworzenie tablicy z krotek
10 a = np.array(((1,2),(3,4)))
11 print("a =", a)
12
13 # tworzenie tablicy zer
14 b = np.zeros(shape=(2,2))
15 print("b =", b)
16
```

```
a = [[1 2]
      [3 4]]
b = [[0. 0.]
      [0. 0.]
```

# Klasa ndarray

Przykładowe sposoby tworzenia tablic

```
numpy_03.py X
17 # tablica trójwymiarowa
18 c = np.zeros(shape=(2,2,2))
19 print("c =", c)
20 print("c.shape =", c.shape)
21 print("c.ndim =", c.ndim)
22 print("c.size =", c.size)
23
24 # tablica jedynek
25 d = np.ones(shape = (10,))
26 print("d =", d)
27 print("d.shape =", d.shape)
28
```

```
c = [[[0. 0.]
      [0. 0.]]
      [[0. 0.]
      [0. 0.]]]
c.shape = (2, 2, 2)
c.ndim = 3
c.size = 8
d = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
d.shape = (10,)
```

# Klasa ndarray

## Przykładowe sposoby tworzenia tablic

```
numpy_04.py X
7  import numpy as np
8
9  # tworzenie wektora jedynek
10 a = np.ones(shape = (3))
11 print("a =", a)
12
13 # utworzenie macirzy diagonalnej
14 b = np.diag(a)
15 print("b =", b)
16
17 # tworzenie macierzy jednostkowej
18 c = np.eye(3)
19 print("c =", c)
20
21
```

```
a = [1. 1. 1.]
b = [[1. 0. 0.]
     [0. 1. 0.]
     [0. 0. 1.]]
c = [[1. 0. 0.]
     [0. 1. 0.]
     [0. 0. 1.]]
```

# Klasa ndarray

Przykładowe sposoby tworzenia tablic

```
numpy_05.py X
6  import numpy as np
7
8  # tworzenie wektora funkcją arange
9  a = np.arange(1,2,.1)
10 print("a =", a)
11
12 # tworzenie wektora funkcją linspace
13 b = np.linspace(1, 2, 11)
14 print("b =", b)
15
16
```

```
Examples/NumPy )
a = [1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
b = [1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```

# Klasa `dtype`

Wszystkie elementy tablicy `ndarray` są tego samego typu

Do reprezentowania typów danych wykorzystuje się klasę `dtype`

Tworząc tablicę funkcją `array` można określić typ danych

# Klasa dtype

```
numpy_06.py X
1  # -*- coding: utf-8 -*-
2  """
3  Tworzenie obiektów ndarray różnych typów.
4  """
5
6  import numpy as np
7
8  # tablica typu int32
9  # (na podstawie typów danych wejściowych)
10 a = np.array((1,2,3))
11 print("a =", a)
12 print("a.dtype =", a.dtype)
13
14 # tablica typu float64
15 # (na podstawie typów danych wejściowych)
16 b = np.array((1,2.,3))
17 print("b =", b)
18 print("b.dtype =", b.dtype)
19
20 # tablica typu float64 (typ domyślny)
21 c = np.zeros(shape = 3)
22 print("c =", c)
23 print("c.dtype =", c.dtype)
```

```
a = [1 2 3]
a.dtype = int32
b = [1. 2. 3.]
b.dtype = float64
c = [0. 0. 0.]
c.dtype = float64
```

# Klasa dtype

```
numpy_06.py X
25 # tablica typu int32 (typ przekazany jako argument)
26 d = np.zeros(shape = 3, dtype = "int32")
27 print("d =", d)
28 print("d.dtype =", d.dtype)
29
30 # tablica typu int8
31 # (liczba 255 nie mieści się w zakresie typu)
32 e = np.array([1, -1, 255], dtype = "int8")
33 print("e =", e)
34 print("e.dtype =", e.dtype)
35
36 # tablica typu uint8
37 # (typ unsigned int nie przechowuje znaku)
38 f = np.array([1, -1, 255], dtype = "uint8")
39 print("f =", f)
40 print("f.dtype =", f.dtype)
41
42 #tablica typu complex128
43 g = np.array([1+1j])
44 print("g =", f)
45 print("g.dtype =", g.dtype)
```

```
d = [0 0 0]
d.dtype = int32
e = [ 1 -1 -1]
e.dtype = int8
f = [  1 255 255]
f.dtype = uint8
g = [  1 255 255]
g.dtype = complex128
```

# Klasa dtype

```
numpy_06.py X
25 # tablica typu int32 (typ przekazany jako argument)
26 d = np.zeros(shape = 3, dtype = "int32")
27 print("d =", d)
28 print("d.dtype =", d.dtype)
29
30 # tablica typu int8
31 # (liczba 255 nie mieści się w zakresie typu)
32 e = np.array([1, -1, 255], dtype = "int8")
33 print("e =", e)
34 print("e.dtype =", e.dtype)
35
36 # tablica typu uint8
37 # (typ unsigned int nie przechowa -1)
38 f = np.array([1, -1, 255], dtype = "uint8")
39 print("f =", f)
40 print("f.dtype =", f.dtype)
41
42 #tablica typu complex128
43 g = np.array([1+1j])
44 print("g =", f)
45 print("g.dtype =", g.dtype)
```

```
d = [0 0 0]
d.dtype = int32
e = [ 1 -1 -1]
e.dtype = int8
f = [  1 255 255]
f.dtype = uint8
g = [  1 255 255]
g.dtype = complex128
```

32: DeprecationWarning: NumPy will stop allowing conversion of out-of-bound Python integers to integer arrays. The conversion of 255 to int8 will fail in the future.

For the old behavior, usually:

```
np.array(value).astype(dtype)
```

will give the desired result (the cast overflows).

```
e = np.array([1, -1, 255], dtype = "int8")
```

38: DeprecationWarning: NumPy will stop allowing conversion of out-of-bound Python integers to integer arrays. The conversion of -1 to uint8 will fail in the future.

For the old behavior, usually:

```
np.array(value).astype(dtype)
```

will give the desired result (the cast overflows).

```
f = np.array([1, -1, 255], dtype = "uint8")
```



# Dostęp do elementów tablic

```
numpy_07.py X
1  """
2  Indeksowanie tablic ndarray.
3  """
4
5  import numpy as np
6
7  #tablica rozmiaru 3x3
8  a = np.arange(1,10,1).reshape(3,3)
9  print("a =", a)
10
11 # indeksowanie elementów tablicy od zera
12 # element 1, 1
13 print("a[1,1] =", a[1,1])
14 print("a[1][1] =", a[1][1])
15 # zerowy wiersz
16 print("a[0] =", a[0])
17 # cała tablica
18 print("a[:]" =, a[:])
```

```
a = [[1 2 3]
      [4 5 6]
      [7 8 9]]
a[1,1] = 5
a[1][1] = 5
a[0] = [1 2 3]
a[:] = [[1 2 3]
        [4 5 6]
        [7 8 9]]
```

# Dostęp do elementów tablic

Dostęp do elementów tablicy można uzyskać używając do indeksowania:

- listy indeksów
- tablicy indeksów
- tablic wartości logicznych

# Dostęp do elementów tablic

```
numpy_09.py X
6 import numpy as np
7
8 #wektor liczb losowych
9 a = np.arange(0,10)
10 print(a)
11
12 #indeksowanie listą
13 index_list = [1, 3, 9]
14 print(a[index_list])
15
16 #indeksowanie tablicą
17 index_array = np.array([0,2,8])
18 print(a[index_array])
19
20 #indeksowanie tablicą logiczną
21 condition = a % 2 == 0
22 print(condition)
23 print(a[condition])
24 print(a[a % 2 == 0])
25
```

```
[0 1 2 3 4 5 6 7 8 9]
[1 3 9]
[0 2 8]
```

```
[ True False  True False  True
 False  True False  True False]
[0 2 4 6 8]
[0 2 4 6 8]
```

# Tworzenie tablic funkcją

```
numpy_08.py X
1  # -*- coding: utf-8 -*-
2  """
3  Tworzenie ndarray z funkcji
4  funkcją fromfunction.
5  """
6
7  import numpy as np
8
9  #tablica jednowymiarowa
10 fun1d = lambda x : x
11 a = np.fromfunction(fun1d, (15, ))
12 print(a)
13
14 #tablica dwuwymiarowa (tabliczka mnożenia)
15 fun2d = lambda x, y : 10*x + y
16 b = np.fromfunction(fun2d, (10, 10), dtype = int)
17 print(b)
18
```

# Tworzenie tablic funkcją

```
numpy_08.py X
1  # -*- coding: utf-8 -*-
2  """
3  Tworzenie ndarray z funkcji
4  funkcją fromfunction.
5  """
6
7  import numpy as np
8
9  #tablica jednowymiarowa
10 fun1d = lambda x : x
11 a = np.fromfunction(fun1d, (15,
12 print(a)
13
14 #tablica dwuwymiarowa (tabliczka)
15 fun2d = lambda x, y : 10*x + y
16 b = np.fromfunction(fun2d, (10,
17 print(b)
18
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14.]
[[ 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 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

# Obliczenia z wykorzystaniem tablic

- Funkcje biblioteki obliczają wartości dla poszczególnych elementów tablic (element po elemencie)

```
numpy_10.py X
1  #-*- coding: utf-8 -*-
2  """
3  Obliczanie wartości funkcji
4  od elementów tablicy.
5  """
6
7  import numpy as np
8
9  a = np.linspace(0, 2*np.pi, 11)
10 cosa = np.cos(a)
11 print("a =", a)
12 print("cos(a) =", cosa)
```

```
numpy
a = [0.          0.62831853 1.25663706
1.88495559 2.51327412 3.14159265
3.76991118 4.39822972 5.02654825
5.65486678 6.28318531]
cos(a) = [ 1.          0.80901699
0.30901699 -0.30901699 -0.80901699 -1.
-0.80901699 -0.30901699 0.30901699
0.80901699 1.          ]
```

# Podsumowanie

Biblioteka `numpy`

Klasa `ndarray`

- tworzenie tablic
- właściwości tablic
- typy danych tablic
- dostęp do elementów tablic
- obliczenia na tablicach