Politechnika Wrocławska

Programowanie obiektowe

INP001129WL

rok akademicki 2022/23

semestr zimowy

Wykład 4

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220

Politechnika Wrocławska

Plan wykładu

- Przeciążanie operatorów
- Przykład
  - __add__
  - __radd__
  - __iadd__
- Metody specjalne klas

Politechnika Wrocławska

Plan wykładu

- Biblioteka **`numpy`**
- Klasa **`ndarray`**
  - tworzenie tablic
  - właściwości tablic
  - typy danych tablic
  - dostęp do elementów tablic
  - obliczenia na tablicach

Politechnika Wrocławska

Przeciążanie operatorów

- Obiekty tworzonych klas mogą wykorzystywać operatory
- Przykładowo, jeśli **`a`** oraz **`b`** są instancjami pewnej klasy, to konieczne jest przeciążenie operatora dodawania tak, aby wyrażenie

  `a + b`

  miało sensowną wartość

Przeciążanie operatorów

- Przykład – dodawanie dwóch punktów

point2d_operators.py

```
"""
Program ilustrujący działanie
operatorem dodwania.
"""
class Point2d_Operators:

    def __init__(self):
        self.__x = 0.
        self.__y = 0.

    def get_coordinates(self):
        return [ self.__x,  s

    def set_xy(self,x,y):
        self.__x = x
        self.__y = y

    # definicja metody __add_
    # przy obliczaniu wyrażen
    def __add__(self, other):
        print("__add__")
        r = Point2d_Operators()
        r.set_xy(self.__x + other.__x, self.__y + other.__y)
        return r

    def __str__(self):
        return str(self.get_coordinates())
```

point2d_21.py

```

def main():
    # zmienna p1 - punkt o współrzędnych [3., 4.]
    p1 = Point2d_Operators()
    p1.set_xy(3.,4.)

    # zmienna p2 - punkt o współrzędnych [2., 0.7]
    p2 = Point2d_Operators()
    p2.set_xy(2.,0.7)

    print( p1 + p2 )

if __name__ == '__main__':
    main()
```

```
__add__
[5.0, 4.7]
```

Politechnika Wrocławska

Przeciążanie operatorów

- Przykład – dodawanie dwóch punktów z obsługą różnych typów

```
34  def main():
35      # zmienna p1 - punkt o współrzędnych [3., 4.]
36      p1 = Point2d_Operators()
37      p1.set_xy(3.,4.)
38
39      # zmienna p2 - punkt o współrzędnych [2., 0.7]
40      p2 = Point2d_Operators()
41      p2.set_xy(2.,0.7)
42
43      print( p1 + p2 )
44      print( p1 + 3. )
45      print( p1 + 3 )
46
```

```
# definicja metody __add__, która jest wywoływana
# przy obliczaniu wyrażenia self + other
def __add__(self, other):
    print("__add__")
    if type(other) == type(self):
        r = Point2d_Operators()
        r.set_xy(self.__x + other.__x, self.__y + other.__y)
        return r
    elif type(other) == int or type(other) == float:
        t = Point2d_Operators()
        t.set_xy(float(other), 0.0)
        return self + t
```

Politechnika Wrocławska

Przeciążanie operatorów

- Przykład – dodawanie dwóch punktów z obsługą różnych typów (zgłaszanie wyjątku)

```
17
18      # definicja metody __add__, która jest wywoływana
19      # przy obliczaniu wyrażenia self + other
20      # zgłasza wyjątek TypeError gdy typ other jest inny niż
21      # Point2d_Operators, int, float
22      def __add__(self, other):
23          print("__add__")
24          if type(other) == type(self):
25              r = Point2d_Operators()
26              r.set_xy(self.__x + other.__x, self.__y + other.__y)
27              return r
28          elif type(other) == int or type(other) == float:
29              t = Point2d_Operators()
30              t.set_xy(float(other), 0.0)
31              return self + t
32          else:
33              raise TypeError
34
```

Przeciążanie operatorów

- Przykład – operator dla różnych typów w zamienionej kolejności

```
41
42  def main():
43      # zmienna p1 - punkt o współrzędnych [3., 4.]
44      p1 = Point2d_Operators()
45      p1.set_xy(3.,4.)
46
47      # zmienna p2 - punkt o współrzędnych [2., 0.7]
48      p2 = Point2d_Operators()
49      p2.set_xy(2.,0.7)
50
51      print( p1 + p2 )
52      print( p1 + 3. )
53      print( p1 + 3 )
54
55      print( 3. + p1 )
56
```

```
34
35      def __radd__(self, other):
36          print("__radd__")
37          return self.__add__(other)
38
```

Politechnika Wrocławska

Przeciążanie operatorów

- Przykład – złożony operator przypisania

```
39      def __iadd__(self, other):
40          print("__iadd__")
41          if type(other) == type(self):
42              self.__x += other.__x
43              self.__y += other.__y
44              return self
45          elif type(other) == int or type(other) == float:
46              self.__x += other
47              return self
48          else:
49              raise TypeError
50
```

Politechnika Wrocławska

Specjalne metody klas

- Klasy mają metody specjalne powiązane z operatorami

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__matmul__(self, other)
object.__truediv__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__divmod__(self, other)
object.__pow__(self, other[, modulo])
object.__lshift__(self, other)
object.__rshift__(self, other)
object.__and__(self, other)
object.__xor__(self, other)
object.__or__(self, other)
```

https://docs.python.org/pl/3/reference/datamodel.html#emulating-numeric-types

Politechnika Wrocławska

Specjalne metody klas

- Podobnie operatory relacji

```
object.__lt__(self, other)
object.__le__(self, other)
object.__eq__(self, other)
object.__ne__(self, other)
object.__gt__(self, other)
object.__ge__(self, other)
```

- Dla wielu innych operacji istnieją metody specjalne (np. :**`__abs__`**, **`__len__`**, **`__repr__`**)

Politechnika Wrocławska

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

Politechnika Wrocławska

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

Politechnika Wrocławska

Klasa `ndarray`

Przykład pokazuje:

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

Politechnika Wrocławska

Klasa ndarray

numpy_01.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Prosty przykład użycia bilioteki numpy
4  """
5
6  # importowanie bilioteki 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 tablice danych
15 npa = np.array(a)
16 npb = np.array(b)
17
18 # biblioteka numpy zawiera definicję kl
19 # reprezentującej tablice danych
20 print(type(npa))
21
22 # tablice ndarray można np. dodawać element po elemencie
23 npc = npa + npb
24 print(npc)
```

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

Politechnika Wrocławska

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

Politechnika Wrocławska

Klasa ndarray

numpy_02.py

```
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 zawierająca 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 elementów tabli
21  print("a.size =", a.size)
22  print("b.size =", b.size)
23
24  # pole ndim - liczba wymiarów
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
```

Politechnika Wrocławska

Klasa ndarray

numpy_02.py

```
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
```

Politechnika Wrocławska

Klasa `ndarray`

Tablice danych można utworzyć na wiele sposobów

numpy_03.py

```python
import numpy as np

# tworzenie tablicy z krotek
a = np.array(((1,2),(3,4)))
print("a =", a)

# tworzenie tablicy zer
b = np.zeros(shape=(2,2))
print("b =", b)

# tablica trójwymiarowa
c = np.zeros(shape=(2,2,2))
print("c =", c)
print("c.shape =", c.shape)
print("c.ndim =", c.ndim)
print("c.size =", c.size)

# tablica jedynek
d = np.ones(shape = (10))
print("d =", d)
print("d.shape =", d.shape)
```

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

Politechnika Wrocławska

Klasa `ndarray`

Tablice danych można utworzyć na wiele sposobów

numpy_04.py

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

```
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`

Tablice danych można utworzyć na wiele sposobów

numpy_05.py

```
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
```

```
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. ]
```

Politechnika Wrocławska

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

https://numpy.org/doc/stable/reference/generated/numpy.dtype.html

Politechnika Wrocławska

Klasa dtype

numpy_06.py

```python
# -*- coding: utf-8 -*-
"""
Tworzenie obiektów ndarray różnych typów.
"""

import numpy as np

# tablica typu int32
# (na podstawie typów danych wejściowych)
a = np.array((1,2,3))
print("a =", a)
print("a.dtype =", a.dtype)

# tablica typu float64
# (na podstawie typów danych wejściowych)
b = np.array((1,2.,3))
print("b =", b)
print("b.dtype =", b.dtype)

# tablica typu float64 (typ domyślny)
c = np.zeros(shape = 3)
print("c =", c)
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
```

Politechnika Wrocławska

Klasa dtype

numpy_06.py

```
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 unisigned 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
```

Politechnika Wrocławska

Dostęp do elementów tablic

numpy_07.py

```
"""
Indeksowanie tablic ndarray.
"""

import numpy as np

#tablica rozmiaru 3x3
a = np.arange(1,10,1).reshape(3,3)
print("a =", a)

# indeksowanie elementów tablicy od zera
# element 1, 1
print("a[1,1] =", a[1,1])
print("a[1][1] =", a[1][1])
# zerowy wiersz
print("a[0] =", a[0])
# cała tablica
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]]
```

Politechnika Wrocławska

Tworzenie tablic funkcją

numpy_08.py

```python
# -*- coding: utf-8 -*-
"""
Tworzenie ndarray z funkcji
funkcją fromfunction.
"""

import numpy as np

#tablica jednowymiarowa
fun1d = lambda x : x
a = np.fromfunction(fun1d, (15, ))
print(a)

#tablica dwuwymiarowa (tabliczka mnożenia)
fun2d = lambda x, y : 10*x + y
b = np.fromfunction(fun2d, (10, 10), dtype = int)
print(b)
```

Politechnika Wrocławska

Tworzenie tablic funkcją

numpy_08.py

```
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
11 a = np
12 print(
13
14 #tabli
15 fun2d
16 b = np
17 print(
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]]
```

Politechnika Wrocławska

Dostęp do elementów tablicy

- Dostęp do elementów tablicy można uzyskać używając do indeksowania:
  - listy indeksów
  - tablicy indeksów
  - tablic wartości logicznych

Politechnika Wrocławska

Dostęp do elementów tablicy

numpy_09.py

```
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 logiczna
21 condition = a % 2 == 0
22 print(condition)
23 print(a[condition])
24 print(a[a % 2 == 0])
```

```
[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]
```

Obliczenia z wykorzystaniem tablic

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

numpy_10.py

```python
# -*- coding: utf-8 -*-
"""
Obliczanie wartości funkcji
od elementów tablicy.
"""

import numpy as np

a = np.linspace(0, 2*np.pi, 11)
cosa = np.cos(a)
print("a =", a)
print("cos(a) =", cosa)
```

```
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.        ]
```

Politechnika Wrocławska

Podsumowanie

- Przeciążanie operatorów
- Metody specjalne klas

- Tablice danych – biblioteka numpy