Politechnika Wrocławska

Programowanie obliczeń komputerowych

W11IKW-SI0106W, W11IKW-SI0106L
rok akademicki 2024/25
semestr zimowy

Wykład 3

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 210

Politechnika Wrocławska

Plan (1)

Klasa a obiekt

Cechy programowania obiektowego

Przykłady

- definicja klasy
- pola i metody
- metody setter/getter
- pola prywatne
- definicja klasy w module

M. Lutz, *Python Wprowadzenie*

Politechnika Wrocławska

Plan (2)

Przeciążanie operatorów

Przykład

- `__add__`
- `__radd__`
- `__iadd__`

Metody specjalne klas

Klasy

- Klasa to ogólny przepis na stworzenie obiektu
- Obiekt (instancja) to pojedynczy egzemplarz danej klasy

Politechnika Wrocławska

Cechy programowania obiektowego

- **abstrakcja**
- **hermetyzacja**
- polimorfizm
- dziedziczenie

Politechnika Wrocławska

Klasy

- Definicja klasy - przykład

point2d_01.py

```
6   # definicja klasy Point2d
7   class Point2d:
8
9       # definicja konstruktora
10      def __init__(self):
11          self.x = 0.
12          self.y = 0.
13
14  def main():
15      # utworzenie instancji klasy
16      my_point = Point2d()
17      # bezpośredni odczyt pól klasy
18      print('współrzędne punktu: [',
19            my_point.x, ', ',
20            my_point.y, ']', sep = '')
21
22  if __name__ == '__main__':
23      main()
```

Politechnika Wrocławska

Klasy

- Definicja klasy - przykład

point2d_01.py

```
6   # definicja klasy Point2d
7   class Point2d:
8
9       # definicja konstruktora
10      def __init__(self):
11          self.x = 0.
12          self.y = 0.
13
14  def main():
15      # utworzenie instancji klasy
16      my_point = Point2d()
17      # bezpośredni odczyt pól klasy
18      print('współrzędne punktu: [',
19            my_point.x, ', ',
20            my_point.y, ']', sep = '')
21
22  if __name__ == '__main__':
23      main()
```

inicjalizacja pól klasy

Politechnika Wrocławska

Klasy

- Klasa zawiera pola i metody

point2d_02.py

```
5   import random
6
7   class Point2d:
8
9       def __init__(self):
10          self.x = 0.
11          self.y = 0.
12
13      #definicja metody random()
14      def random(self):
15          """
16          Meotda nadaje współrzędnym punktu
17          losowe wartości
18
19          """
20          self.x = random.uniform(0,1)
21          self.y = random.uniform(0,1)
22
```

definicja metody

Politechnika Wrocławska

Klasy

- Wywołanie metody

```
22
23  def main():
24      my_point = Point2d()
25      print('współrzędne punktu: [',
26            my_point.x, ', ', my_point.y, ']', sep = '')
27      #wywołanie metody
28      my_point.random()
29      print('współrzędne punktu: [',
30            my_point.x, ', ', my_point.y, ']', sep = '')
31
```

Politechnika Wrocławska

Klasy

- Do nadawania wartości polom klasy wykorzystuje się odpowiednie metody („setter”/„getter”)

point2d_04.py

```
5   import random
6
7   class Point2d:
8
9       def __init__(self):
10          self.x = 0.
11          self.y = 0.
12
13      def random(self):
14          self.x = random.uniform(0,1)
15          self.y = random.uniform(0,1)
16
17      #definicja metody
18      def get_coordinates(self):
19          return [self.x, self.y]
20
```

odczytywanie wartości pól (getter)

Politechnika Wrocławska

Klasy

- Do nadawania wartości polom klasy wykorzystuje się odpowiednie metody („setter”/„getter”)

```
21  def main():
22      my_point = Point2d()
23      #wywołanie metody get_coordinates()
24      print('współrzędne punktu: ',
25            my_point.get_coordinates(), sep = '')
```

odczytywanie wartości pól (getter)

Politechnika Wrocławska

Klasy

- Nazwy pól prywatnych oznacza się dwoma podkreślnikami

point2d_05.py

```
4   import random
5
6   class Point2d:
7
8       def __init__(self):
9           #inicjalizacja pól prywatnych
10          self.__x = 0.
11          self.__y = 0.
12
13      def random(self):
14          self.__x = random.uniform(0,1)
15          self.__y = random.uniform(0,1)
16
17      def get_coordinates(self):
18          return [self.__x, self.__y]
```

Politechnika Wrocławska

Klasy

- Nazwy pól prywatnych oznacza się dwoma podkreślnikami

point2d_05.py | point2d_06.py

```
20  def main():
21      # dane w klasie obsługiwane
22      # są tylko za pomocą metod
23      my_point = Point2d()
24      print('współrzędne punktu: ',
25            my_point.get_coordinates(), sep = '')
26      my_point.random()
27      print('współrzędne punktu: ',
28            my_point.get_coordinates(), sep = '')
29      # poniższa instrukcja powoduje błąd
30      # print(my_point.__x)
31
32
33  if __name__ == '__main__':
34      main()
35
```

Politechnika Wrocławska

Klasy

- Nazwy pól prywatnych oznacza się dwoma podkreślnikami

point2d_06.py

```
19  def main():
20      my_point = Point2d()
21      print('współrzędne punktu: ',
22            my_point.get_coordinates(), sep = '')
23      my_point.random()
24      # poniższe instrukcje nie dają dostępu do
25      # prywatnych pól klasy
26      my_point.__x = .5
27      my_point.__y = .5
28      print('współrzędne punktu: ',
29            my_point.get_coordinates(), sep = '')
30
```

Politechnika Wrocławska

Klasy

- Przykład metody nadającej wartości polom klasy

point2d_07.py

```
5   import random
6
7   class Point2d:
8
9       def __init__(self):
10          self.__x = 0.
11          self.__y = 0.
12
13      def random(self):
14          self.__x = random.uniform(0,1)
15          self.__y = random.uniform(0,1)
16
17      def get_coordinates(self):
18          return [self.__x, self.__y]
19
20      #definicja metody
21      def set_xy(self,x,y):
22          self.__x = x
23          self.__y = y
```

nadawanie wartości pól (setter)

Politechnika Wrocławska

Klasy

- Przykład metody nadającej wartości polom klasy

point2d_07.py

```
24
25    def main():
26        my_point = Point2d()
27        print('współrzędne punktu: ',
28              my_point.get_coordinates(), sep = '')
29        #wywołanie metody
30        my_point.set_xy(.5,.5)
31        print('współrzędne punktu: ',
32              my_point.get_coordinates(), sep = '')
```

nadawanie wartości pól (setter)

Politechnika Wrocławska

Klasy

- Wykorzystanie metody klasy przez inne metody klasy

point2d_08.py

```
5   import random
6
7   class Point2d:
8
9       def __init__(self):
10          self.__x = 0.
11          self.__y = 0.
12
13      #metoda random wywołuje metodę set_xy
14      def random(self):
15          self.set_xy(random.uniform(0,1), random.uniform(0,1))
16
17      def get_coordinates(self):
18          return [self.__x, self.__y]
19
20      def set_xy(self,x, y):
21          self.__x = x
22          self.__y = y
```

Politechnika Wrocławska

Klasy

- Definicja klasy może być umieszczona w osobnym module

point2d.py | point2d_09.py

```
4   import random
5
6   class Point2d:
7
8       def __init__(self):
9           self.__x = 0.
10          self.__y = 0.
11
12      def random(self):
13          self.set_xy(random.uniform(0,1),
14
15      def get_coordinates(self):
16          return [ self.__x,  self.__y ]
17
18      def set_xy(self,x,y):
19          self.__x = x
20          self.__y = y
21
22      #funkcja implementuje konwersję obiektu na string
23      def __str__(self):
24          return str(self.get_coordinates())
25
```

point2d.py | point2d_09.py

```
5   import point2d
6
7   def main():
8       my_point = point2d.Point2d()
9       # obiekt klasy jako argument funkcji print
10      # do wyświetlenia wykorzystywana jest metoda __str__()
11      print('współrzędne punktu: ', my_point)
12      my_point.set_xy(.5, .5)
13      print('współrzędne punktu: ', my_point)
14
15
16  if __name__ == '__main__':
17      main()
18
19
```

Politechnika Wrocławska

Przeciążanie operatorów

- Obiekty tworzonych klas mogą wykorzystywać operatory
- Przykładowo, jeśli **`a`** oraz **`b`** są instancjami pewnej klasy, to, aby wyrażenie

  `a + b`

  miało sensowną wartość, konieczne jest przeciążenie operatora dodawania

Politechnika Wrocławska

Przeciążanie operatorów

Przykład - dodawanie dwóch punktów

point2d_operators.py | point2d_21.py

```
"""
Program ilustrujący działanie klasy z przeciążonym
operatorem dodwania.
"""
class Point2d_Operators:

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

    def get_coordinates(self):
        return [ self.__x,  self.__y ]

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

    # definicja metody __add__, która jest wywoływana
    # przy obliczaniu wyrażenia self + other
    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_operators.py | 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 - obsługa 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 - 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
```

Politechnika Wrocławska

Przeciążanie operatorów

Przykład - różne typy w odwróconej 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

Przeciążanie operatorów

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

Przeciążanie operatorów

Metody specjalne powiązane z operatorami 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)
```

Wybrane metody specjalne

```
__abs__
__len__
__repr__
```

Podsumowanie

- Klasa a obiekt
- Cechy programowania obiektowego
- Przeciążanie operatorów
- Metody specjalne klas