Politechnika Wrocławska

Programowanie obiektowe

INP001129WL

rok akademicki 2022/23

semestr zimowy

Wykład 3

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220

Politechnika Wrocławska

Plan wykładu (1)

- Wyrażenia listowe
- Wyrażenia generujące
- Funkcje anonimowe
- Domknięcie

M. Lutz, *Python Wprowadzenie*

T. Gaddis, *Python dla zupełnie początkujących*

Politechnika Wrocławska

Plan wykładu (2)

- Klasy
  - pola klas
  - metody klas
  - metody dostępu do pól
  - pola prywatne

M. Lutz, *Python Wprowadzenie*

T. Gaddis, *Python dla zupełnie początkujących*

Wyrażenie listowe

- Wyrażenie listowe można wykorzystać do generowania listy list

comprehension_list_04.py

```
# -*- coding: utf-8 -*-
"""
Przykłady użycia wyrażenia listowego
do utworzenia listy list.
"""

matrix = [[i + j for j in range(5) ] for i in range(7)]
print(matrix)

matrix2 = [["*" for j in range(i)] for i in range(5)]
print(matrix2)
```

```
[[0, 1, 2, 3, 4], [1, 2, 3, 4, 5], [2, 3, 4, 5, 6], [3, 4, 5, 6, 7],
[4, 5, 6, 7, 8], [5, 6, 7, 8, 9], [6, 7, 8, 9, 10]]
[[], ['*'], ['*', '*'], ['*', '*', '*'], ['*', '*', '*', '*']]
```

Politechnika Wrocławska

Wyrażenie generujące

- Wyrażenie generujące może być zastosowane, gdy wszystkie elementy listy nie muszą być znane jednocześnie
- Wyrażenie generujące objęte jest nawiasami zwykłymi ()

Politechnika Wrocławska

Wyrażenie generujące

- Przykład

generator_expressions_01.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Przykładowy skrypt pozwalający porównać
4   wyrażenia listowe oraz wyrażenia generujące
5   """
6
7   #lista kwadratów liczb
8   list1 = [x*x for x in range(10)]
9   print(list1)
10  #suma elementów tej listy
11  print(sum(list1))
12
13  #suma elementów listy utworzonej przez wyrażenie listowe
14  print(sum([x*x for x in range(10)]))
15
16  #suma obliczona z wykorzystaniem wyrażenia generującego
17  print(sum((x*x for x in range(10))))
18
19  #wyrażenie generujące
20  g = (x*x for x in range(10))
21
```

Politechnika Wrocławska

Wyrażenie generujące

- Przykład

generator_expressions_02.py

```
# -*- coding: utf-8 -*-
"""
Przykład pokazujący działanie generatora
utworzonego przez wyrażenie generujące.
"""

#utworzenie generatora z wykorzystaniem wyrażenia generującego
g = (x*x for x in range(3))

#odczytanie wartości generatora
print(next(g))
input()

#kolejne odczytanie wartości generatora
print(next(g))
input()

#kolejne odczytanie wartości generatora
print(next(g))
input()

#to odczytanie wartości generatora skutkuje błędem
print(next(g))
```

Wyrażenie generujące

- Przykład

generator_expressions_02.py

```
1   # -*- coding: utf-8
2   """
3   Przykład pokazujący
4   utworzonego przez wy
5   """
6
7   #utworzenie generatc
8   g = (x*x for x in ra
9
10  #odczytanie wartości
11  print(next(g))
12  input()
13
14  #kolejne odczytanie
15  print(next(g))
16  input()
17
18  #kolejne odczytanie
19  print(next(g))
20  input()
21
22  #to odczytanie wartości generatora skutkuje błędem
23  print(next(g))
```

```
0


1


4


Traceback (most recent call last):

  File "D:\repozytoria\python-examples\generatory
\generator_expressions_02.py", line 23, in <module>
    print(next(g))

StopIteration
```

Politechnika Wrocławska

Funkcje anonimowe

- Mechanizm funkcji anonimowych pozwala zwięźle zapisać kod małych funkcji
- Definicja funkcji może zawierać pojedyncze wyrażenie
- Definicja funkcji anonimowej wykorzystuje słowo kluczowe **`lambda`**

Politechnika Wrocławska

Funkcje anonimowe

- Przykład

lambda_01.py

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazujący użycie funkcji anonimowej.
"""

# definicja funkcji anonimowej (funkcja przypisana do zmiennej f)
f = lambda x : x*x

# wywołanie funkcji anonimowej (przechowywanej w zmiennej f)
for i in range(10):
    print(f(i))

# funkcja anonimowa zdefiniowana w miejscu wywołania
for i in range(10):
    print((lambda x : x*x)(i))
```

Politechnika Wrocławska

Funkcje anonimowe

- Przykład

lambda_02.py

```
# -*- coding: utf-8 -*-
"""
Przykład pokazujący wykorzystanie funkcji anonimowej
przyjmującej dwa argumenty.
"""

# definicja funkcji anonimowej (funkcja przypisana do zmiennej f)
multipler = lambda x,y : x*y

# wywołanie funkcji anonimowej (przechowywanej w zmiennej f)
for i in range(10):
    print(multipler(i,2))
```

Politechnika Wrocławska

Funkcje anonimowe

- Przykład

lambda_03.py

```
# -*- coding: utf-8 -*-
"""
Przykład pokazujący funkcję anonimową
wykorzystujący instrukcję warunkową.
"""

# definicja funkcji anonimowej
# (funkcja przypisana do zmiennej smaller)
smaller = lambda x,y : x if x < y else y

# wywołanie funkcji anonimowej
for i in [1, 3, 5, 7, 4, 8, 2, 6]:
    print(smaller(i,5))
```

Politechnika Wrocławska

Domknięcie

- Obiekt funkcji może być powiązany ze stanem otaczających go zasięgów
- Funkcję z pamięcią stanu nazywamy domknięciem

Politechnika Wrocławska

Domknięcie

- Przykład

closure_01.py

```
# -*- coding: utf-8 -*-
"""
Przykład pokazujący wykorzystanie domknięcia.
"""

# funkcja multiplication przyjmuje mnożnik
# zwraca funkcję, która przyjmuje mnożną
# i oblicza wynik mnożenia
def multiplication(multiplier):
    def multipy(multiplicant):
        return multiplicant*multiplier

    return multipy

# funkcja multiplication dla mnożnika 2 zwraca
# funkcję podwającą
doubler = multiplication(2)
print(doubler(4))
```

Politechnika Wrocławska

Domknięcie

- Przykład

```
15  # funkcja multiplication dla mnożnika 2 zwraca
16  # funkcję podwajającą
17  doubler = multiplication(2)
18  print(doubler(4))
19
20  # funkcja multiplication dla mnożnika 2 zwraca
21  # funkcję potrajającą
22  tripler = multiplication(3)
23  print(tripler(7))
```

Politechnika Wrocławska

Domknięcie

- Przykład z wykorzystaniem funkcji anonimowej

closure_02.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Przykład pokazujący wykorzystanie domknięcia.
4   """
5
6   # funkcja multiplication przyjmuje mnożnik
7   # zwraca funkcję, która przyjmuje mnożną
8   # i oblicza wynik mnożenia
9   def multiplication(multiplier):
10      return lambda multiplicant: multiplicant*multiplier
11
12  # funkcja multiplication dla mnożnika 2 zwraca
13  # funkcję podwającą
14  doubler = multiplication(2)
15  print(doubler(4))
```

Politechnika Wrocławska

Klasy

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

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
```

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)

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)

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
```

Klasy

- Definicja klasy może być umieszczona w osobnym module

point2d.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(
14
15      def get_coordinates(self):
16          return [ self.__x,  self.__
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_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

Klasy

- Cechy programowania obiektowego
  - **abstrakcja**
  - **hermetyzacja**
  - polimorfizm
  - dziedziczenie

Politechnika Wrocławska

Podsumowanie

- Wyrażenia listowe
- Wyrażenia generujące
- Funkcje anonimowe
- Domknięcie

- Klasy