



Politechnika
Wrocławska

Programowanie obiektowe

W11FTE-SI0080G (INP001129WL)

rok akademicki 2023/24

semestr letni

Wykład 3

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220



Plan

Wyrażenia listowe

Wyrażenia generujące

Funkcje anonimowe

Domknięcie

Wyrażenie listowe

- Wyrażenia listowe (comprehension list) pozwalają w łatwy sposób generować listy
- W wielu przypadkach wyrażenie listowe może zastąpić pętlę for

Wyrażenie listowe

- Przykłady

```
comprehension_list.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład użycia wyrażenia listowego
4  (comprehension list, listy składanej)
5  do utworzenia listy.
6  """
7
8  square = []
9  for x in range(5):
10     square += [x*x]
11
12  print(square)
13
14  square2 = [x*x for x in range(5)]
15  print(square2)
16
```

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

Wyrażenie listowe

- Przykłady

```
comprehension_list_02.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład użycia wyrażenia listowego
4  (comprehension list, listy składanej)
5  do utworzenia listy, z wykorzystaniem instrukcji warunkowej.
6  """
7
8  square = []
9  for x in range(5):
10     if x % 2 == 0:
11         square += [x*x]
12
13  print(square)
14
15  square2 = [x*x for x in range(5) if x % 2 == 0]
16  print(square2)
17
```

```
[0, 4, 16]
[0, 4, 16]
```

Wyrażenie listowe

- Przykłady

```
comprehension_list_03.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykłady użycia wyrażenia listowego
4  (comprehension list, listy składanej).
5
6  Przykład pochodzi ze strony internetowej:
7  https://www.w3schools.com/python/python\_lists\_comprehension.asp
8  """
9
10 fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
11
12 newlist = [x for x in fruits if "a" in x]
13 print(newlist)
14 input()
15
16 newlist = [x for x in fruits if x != "apple"]
17 print(newlist)
18 input()
```

Wyrażenie listowe

- Przykłady

```
comprehension_list_03.py X
16 newlist = [x for x in fruits if x != "apple"]
17 print(newlist)
18 input()
19
20 newlist = [x for x in fruits]
21 print(newlist)
22 input()
23
24 newlist = [x.upper() for x in fruits]
25 print(newlist)
26 input()
27
28 newlist = ['hello' for x in fruits]
29 print(newlist)
30 input()
31
32 newlist = [x if x != "banana" else "orange" for x in fruits]
33 print(newlist)
34 input()
```

Wyrażenie listowe

- Wyrażenie listowe można wykorzystać do generowania listy list

```
comprehension_list_04.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykłady użycia wyrażenia listowego
4  do utworzenia listy list.
5  """
6
7  matrix = [[i + j for j in range(5) ] for i in range(7)]
8  print(matrix)
9
10 matrix2 = [["*" for j in range(i)] for i in range(5)]
11 print(matrix2)
12
```


Wyrażenie listowe

- Wyrażenie listowe można wykorzystać do generowania listy list

```
comprehension_list_04.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykłady użycia wyrażenia listowego
4  do utworzenia listy list.
5  """
6
7  matrix = [[i + j for j in range(5) ] for i in range(7)]
8  print(matrix)
9
10 matrix2 = [["*"] for j in range(i)] for i in range(5)]
11 print(matrix2)
12
```

```
[[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]]
[[], ['*'], ['*', '*'], ['*', '*', '*'], ['*', '*', '*', '*']]
```

Wyrażenie generujące

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

Wyrażenie generujące

- Przykład

```
generator_expressions_01.py X
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
```

Wyrażenie generujące

- Przykład

```
generator_expressions_02.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący działanie generatora
4  utworzonego przez wyrażenie generujące.
5  """
6
7  #utworzenie generatora z wykorzystaniem wyrażenia generującego
8  g = (x*x for x in range(3))
9
10 #odczytanie wartości generatora
11 print(next(g))
12 input()
13
14 #kolejne odczytanie wartości generatora
15 print(next(g))
16 input()
17
18 #kolejne odczytanie wartości generatora
19 print(next(g))
20 input()
21
22 #to odczytanie wartości generatora skutkuje błędem
23 print(next(g))
24
```

Wyrażenie generujące

- Przykład

```
generator_expressions_02.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący działanie generatora
4  utworzonego przez wyrażenie generujące.
5  """
6
7  #utworzenie generatora z wykorzystaniem wyrażeni
8  g = (x*x for x in range(3))
9
10 #odczytanie wartości generatora
11 print(next(g))
12 input()
13
14 #kolejne odczytanie wartości generatora
15 print(next(g))
16 input()
17
18 #kolejne odczytanie wartości generatora
19 print(next(g))
20 input()
21
22 #to odczytanie wartości generatora skutkuje błędem
23 print(next(g))
24
```

```
0
1
4
Traceback (most recent call last):
  File "D:\repozytoria\python-examples\generator
\generator_expressions_02.py", line 23, in <module>
    print(next(g))
StopIteration
```

Funkcje anonimowe

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

Funkcje anonimowe

- Przykład

```
lambda_01.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący użycie funkcji anonimowej.
4  """
5
6  # definicja funkcji anonimowej (funkcja przypisana do zmiennej f)
7  f = lambda x : x*x
8
9  # wywołanie funkcji anonimowej (przechowywanej w zmiennej f)
10 for i in range(10):
11     print(f(i))
12
13 # funkcja anonimowa zdefiniowana w miejscu wywołania
14 for i in range(10):
15     print((lambda x : x*x)(i))
```

Funkcje anonimowe

- Przykład

```
lambda_02.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący wykorzystanie funkcji anonimowej
4  przyjmującej dwa argumenty.
5  """
6
7  # definicja funkcji anonimowej (funkcja przypisana do zmiennej f)
8  multiplier = lambda x,y : x*y
9
10 # wywołanie funkcji anonimowej (przechowywanej w zmiennej f)
11 for i in range(10):
12     print(multiplier(i,2))
13
```


Funkcje anonimowe

- Przykład

```
lambda_03.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący funkcję anonimową
4  wykorzystujący instrukcję warunkową.
5  """
6
7  # definicja funkcji anonimowej
8  # (funkcja przypisana do zmiennej smaller)
9  smaller = lambda x,y : x if x < y else y
10
11 # wywołanie funkcji anonimowej
12 for i in [1, 3, 5, 7, 4, 8, 2, 6]:
13     print(smaller(i,5))
14
```

Domknięcie

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

Domknięcie

- Przykład

```
closure_01.py X
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     def multiply(multiplicand):
11         return multiplicand*multiplier
12
13     return multiply
14
15 # funkcja multiplication dla mnożnika 2 zwraca
16 # funkcję podwajającą
17 doubler = multiplication(2)
18 print(doubler(4))
```

Domknięcie

- Przykład

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

Domknięcie

- Przykład z wykorzystaniem funkcji anonimowej

```
closure_02.py X
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 multiplicand: multiplicand*multiplier
11
12 # funkcja multiplication dla mnożnika 2 zwraca
13 # funkcję podwajającą
14 doubler = multiplication(2)
15 print(doubler(4))
```

Podsumowanie

Wyrażenia listowe

Wyrażenia generujące

Funkcje anonimowe

Domknięcie