



Politechnika
Wrocławska

Programowanie obiektowe

W11FTE-SI0080G (INP001129WL)

rok akademicki 2023/24

semestr letni

Wykład 2

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220



Plan

Typy danych

- Przypomnienie poznanych typów prostych i złożonych
- NoneType

Przekazywanie argumentów

- argumenty formalne i faktyczne
- argumenty pozycyjne i nazwane
- argumenty domyślne

Zwracanie wyników

Typy danych

- Python wykorzystuje dynamiczną kontrolę typów
- W przykładzie, w kolejnych wywołaniach funkcji `show_data_type()` argument jest innego typu

```
data_types.py X
1  # -*- coding: utf-8 -*-
2  """
3  Program pokazuje dynamiczną kontrolę typów.
4  """
5
6  def main():
7      # typy liczbowe
8      show_data_type(1)
9      show_data_type(1.0)
10     show_data_type(1j)
11
```

Typy danych

```
11
12     # inne typy proste
13     show_data_type(True)
14     show_data_type("napis")
15     show_data_type(None)
16
17     # typy złożone
18     show_data_type([1, 2])
19     show_data_type((1.0, 0.0))
20     show_data_type({'k': 'v'})
21     show_data_type({'k'})
22
```

```
24     # funkcja wyświetla zawartość argumentu
25     # oraz wyświetla informacje o jego typie
26     # wywołanie funkcji type()
27     def show_data_type(x):
28         print(x, "jest typu", type(x))
29
30
31     if __name__ == "__main__":
32         main()
33
```

Typy danych

```
1 jest typu <class 'int'>  
1.0 jest typu <class 'float'>  
1j jest typu <class 'complex'>  
True jest typu <class 'bool'>  
napis jest typu <class 'str'>  
None jest typu <class 'NoneType'>  
[1, 2] jest typu <class 'list'>  
(1.0, 0.0) jest typu <class 'tuple'>  
{'k': 'v'} jest typu <class 'dict'>  
{'k'} jest typu <class 'set'>
```

Typy danych - `NoneType`

- Specjalny typ danych wykorzystywany, gdy zmienna nie ma przypisanej wartości
- Posiada tylko jedną wartość `None`
- Nie może być wykorzystywany w wyrażeniach

Typy danych - NoneType

- Przykład: funkcja `fun()` nie zwraca wartości

```
none_type.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje wykorzystanie typu NoneType,
4  gdy funkcja nie zwraca żadnej wartości.
5  """
6
7  def main():
8      #wypisanie wartości zwracanej przez funkcję fun()
9      print(fun())
10
11 # funkcja fun() nie przyjmuje żadnych argumentów
12 # nie zwraca także żadnej wartości (brak instrukcji return)
13 # wartością zwracaną przez funkcję jest None (typu NoneType)
14 def fun():
15     print("Komunikat z funkcji fun()")
16
17 if __name__ == "__main__":
18     main()
```

Typy danych - NoneType

- Przykład: funkcja `fun()` nie zwraca wartości

```
none_type.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje wykorzystanie typu NoneType,
4  gdy funkcja nie zwraca żadnej wartości.
5  """
6
7  def main():
8      #wypisanie wartości zwracanej przez fun
9      print(fun())
10
11 # funkcja fun() nie przyjmuje żadnych argumentów
12 # nie zwraca także żadnej wartości (brak instrukcji return)
13 # wartością zwracaną przez funkcję jest None (typu NoneType)
14 def fun():
15     print("Komunikat z funkcji fun()")
16
17 if __name__ == "__main__":
18     main()
```

Komunikat z funkcji fun()
None

Przekazywanie argumentów

- Przypomnienie:
 - lista argumentów funkcji (umieszczona w nagłówku funkcji) nazywana jest listą argumentów formalnych (formal arguments)
 - wartości wyrażeń umieszczone w wywołaniu funkcji nazywane argumentami faktycznymi (actual arguments)
 - argumenty do funkcji można przekazywać jako argument nazwane lub argumenty pozycyjne
 - argumenty nazwane następują po argumentach pozycyjnych

Przekazywanie argumentów

- Argumenty domyślne to argumenty formalne, którym nadajemy wartość początkową, dzięki temu mogą być pomijane przy wywołaniu funkcji
- Operatory * oraz ** mogą być wykorzystywane do przekazywania do funkcji listy argumentów faktycznych o zmiennej długości

Przekazywanie argumentów

- Argumenty, które są niemodyfikowalne (liczby, krotki itd.) są przekazywane przez wartość
- Argumenty, które są modyfikowalne (listy, słowniki) są przekazywane przez referencję

Przekazywanie argumentów

- Przykład

```
arguments_01.py X arguments_02.py X arguments_03.py X arguments_04.py X
1 # -*- coding: utf-8 -*-
2 """
3 Przykład ilustruje przekazanie argumentu przez wartość
4 """
5
6 def f(a):
7     a = 99
8
9     b = 88
10    f(b)
11    print("b =", b)
12    # zmienna b ma wartość 88 mimo wywołania funkcji f(b)
13
```

Przekazywanie argumentów

- Przykład


```
arguments_02.py X arguments_03.py X arguments_04.py X arg
1 # -*- coding: utf-8 -*-
2 """
3 Przykład ilustruje przekazanie argumentów
4 przez wartość oraz przez referencję
5 """
6
7 def f(a, b):
8     a = 2
9     b[0] = 'python'
10
11 x = 1
12 l = [1, 2]
13 f(x, l)
14 print(x, l)
15
16
```

```
1 ['python', 2]
```

Przekazywanie argumentów

- Przykład

```
arguments_03.py X arguments_04.py X arguments_05.py X arguments_06.  
1 # -*- coding: utf-8 -*-  
2 """  
3 Przekazanie argumentów jako argumenty pozycyjne  
4 """  
5  
6 def f(a,b):  
7     print(a, b)  
8  
9 x = 1  
10 y = 2  
11 f(x,y)  
12
```



Przekazywanie argumentów

- Przykład

```
arguments_04.py X arguments_05.py X arguments_06.py X arguments_
1 # -*- coding: utf-8 -*-
2 """
3 Przekazanie argumentów jako argumenty nazwane
4 """
5
6 def f(a,b):
7     print(a, b)
8
9 f(b = 1, a = 2)
10
```

Przekazywanie argumentów

- Przykład

```
arguments_05.py X arguments_06.py X arguments_07.py X argu
1 # -*- coding: utf-8 -*-
2 """
3 Definicja funkcji z argumentem domyślnym
4 """
5
6 def f(a, b = 0):
7     print(a, b)
8
```


Przekazywanie argumentów

- Przykład

```
arguments_05.py X arguments_06.py X
1 # -*- coding: utf-8 -*-
2 """
3 Definicja funkcji z argume
4 """
5
6 def f(a, b = 0):
7     print(a, b)
8
```

```
9 # wartość a dopasowane pozycyjnie
10 # wartość b nadana domyślna
11 f(1)
12
13 # oba argumenty nadane pozycyjnie
14 # (zastąpiona wartość domyślna)
15 f(2,3)
16
17 # wartość a dopasowane pozycyjnie
18 # wartość b nadana jako argument nazwany
19 f(4,b=5)
20
21 # oba argumenty jako argumenty nazwane
22 f(b=6,a=7)
23
```

Przekazywanie argumentów

- Przykład

```
arguments_05.py X arguments_06.py X
1 # -*- coding: utf-8 -*-
2 """
3 Definicja funkcji z argume
4 """
5
6 def f(a, b = 0):
7     print(a, b)
8
```

```
9 # wartość a dopasowane pozycyjnie
10 # wartość b nadana domyślna
11 f(1)
12
13 # oba argumenty nadane pozycyjnie:
14 # (zastąpiona wartość domyślna)
15 f(2,3)
16
17 # wartość a dopasowane pozycyjnie:
18 # wartość b nadana jako argument
19 f(4,b=5)
20
21 # oba argumenty jako argumenty nazwane
22 f(b=6,a=7)
23
```

| | |
|---|---|
| 1 | 0 |
| 2 | 3 |
| 4 | 5 |
| 7 | 6 |

Przekazywanie argumentów

- Przykład

```
arguments_07.py X arguments_08.py X arguments_09.py X
1  # -*- coding: utf-8 -*-
2  """
3  Definicja funkcji przyjmującej wiele argumentów
4  (operator *)
5  """
6
7  def f(*a):
8      for x in a:
9          print(x)
10
11
12  f(1)
13
14  print()
15
16  f(2,3,4,5)
17
```

```
1
2
3
4
5
```

Przekazywanie argumentów

- Przykład

```
arguments_08.py X arguments_09.py X
1  # -*- coding: utf-8 -*-
2  """
3  Definicja funkcji przyjmującej wiele argumentów nazwanych
4  (operator **)
5  """
6
7  def f(**a):
8      for k, v in a.items():
9          print(k, v)
10
11
12  f(a=1)
13
14  print()
15
16  f(b=2, c=3, d=4, e=5)
17
18
```

| | |
|---|---|
| a | 1 |
| b | 2 |
| c | 3 |
| d | 4 |
| e | 5 |

Przekazywanie argumentów

- Przykład

```
arguments_09.py X
1  # -*- coding: utf-8 -*-
2  """
3  Definicja funkcji, w której opcjonalny argument
4  ma domyślną wartość None
5  """
6
7  def f(a, b = None):
8      if b is None:
9          print('Wywołanie z jednym argumentem', a)
10     else:
11         print('Wywołanie z dwoma argumentami', a, b)
12
13
14
15  f(1)
16  f(2,3)
17
```

Wywołanie z jednym argumentem 1
Wywołanie z dwoma argumentami 2 3

Zwracanie wyniku

- Funkcja może zwracać krotkę, zawierającą kilka wartości, w ten sposób funkcja może zwrócić więcej niż jedną wartość

Zwracanie wyniku

- Przykład

```
return_result_04.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykładowa funkcja zwracająca trzy wyniki (zapakowane w krotkę).
4  """
5
6  def fun(x,y):
7      return x+y, x-y, y-x
8
9  a = 4
10 b = 7
11
12 s, ra, rb = fun(a,b) #rozpakowanie krotki do trzech zmiennych
13 print("a + b = ", s)
14 print("a - b = ", ra)
15 print("b - a = ", rb)
16
```

```
a + b = 11
a - b = -3
b - a = 3
```

Zwracanie wyniku

- Przykład

```
return_result_05.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykładowa funkcja zwracająca trzy wyniki (zapakowane w krotkę).
4  Dwie pierwsze wartości są ignorowane przy przypisaniu.
5  """
6
7  def fun(x,y):
8      return x+y, x-y, y-x
9
10 a = 4
11 b = 7
12
13 _, _, rb = fun(a,b)
14 #print("a + b = ", s)
15 #print("a - b = ", ra)
16 print("b - a = ", rb)
```


Podsumowanie

Typy danych

- Przypomnienie poznanych typów prostych i złożonych
- NoneType

Przekazywanie argumentów

- argumenty formalne i faktyczne
- argumenty pozycyjne i nazwane
- argumenty domyślne

Zwracanie wyników