Politechnika Wrocławska

# Podstawy programowania W11OPA-SI0072G
# Wstęp do programowania W11FTE-SI0141WL
# Wstęp do programowania W11IKW-SI0080WL

rok akademicki 2023/24
semestr letni

# Wykład 4


Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220


Politechnika Wrocławska

# Plan prezentacji (1)

- Wprowadzenie do list
- Operator powtórzenia
- Listy i pętla `for`
- Indeksowanie elementów
- Długość listy

T. Gaddis, *Python dla zupełnie początkujących*, Helion, 2019

Politechnika Wrocławska

# Plan prezentacji (2)

- Modyfikowanie list
- Konkatenacja list
- Wycinek listy
- Wyszukiwanie elementu listy
- Wybrane metody listy
- Kopiowanie listy
- Przetwarzanie danych
- Listy zagnieżdżone

T. Gaddis, *Python dla zupełnie początkujących*, Helion, 2019

Politechnika Wrocławska

# Plan prezentacji (3)

- Krotki (niemodyfikowalne sekwencje)
- Rozpakowywanie krotek

- Zmienna liczba argumentów wywołania funkcji

T. Gaddis, *Python dla zupełnie początkujących*, Helion, 2019

Politechnika Wrocławska

# Wprowadzenie do list

- Lista to obiekt zawierający wiele elementów danych.
- Lista jest modyfikowalna – jej zawartość może być zmieniona w trakcie działania programu.

Politechnika Wrocławska

# Wprowadzenie do list

- Przykład polecenia tworzącego listę:
  **`parzyste = [2, 4, 6, 8, 10]`**
- Listę można wyświetlić funkcją **`print()`**
  **`print(parzyste)`**

```
In [1]: parzyste = [2, 4, 6, 8, 10]

In [2]: print(parzyste)
[2, 4, 6, 8, 10]
```

Politechnika Wrocławska

# Wprowadzenie do list

- Lista może zawierać ciągi tekstowe:

```
kursy = ['Algebra', 'Analiza', 'Fizyka']
```

```
In [3]: kursy = ['Algebra', 'Analiza', 'Fizyka']

In [4]: print(kursy)
['Algebra', 'Analiza', 'Fizyka']
```

Politechnika Wrocławska

# Wprowadzenie do list

- Lista może zawierać elementy o różnych typach danych:

```
zawodnik = ['Bartosz Zmarzlik',78,2.410]
```

```
In [5]: zawodnik = ['Bartosz Zmarzlik', 78, 2.410]

In [6]: print(zawodnik)
['Bartosz Zmarzlik', 78, 2.41]
```

Politechnika Wrocławska

# Wprowadzenie do list

- Istnieje wbudowana funkcja **`list()`** przeznaczona do konwertowania różnych typów obiektów na postać listy

  **`liczby = list(range(5))`**

```
In [8]: print(range (5))
range(0, 5)

In [9]: liczby = list(range(5))

In [10]: print(liczby)
[0, 1, 2, 3, 4]
```

Politechnika Wrocławska

# Operator powtórzenia

- Operator `*` zastosowany do listy i liczby całkowitej nazywany jest operatorem powtórzenia
- Operator powtórzenia powoduje utworzenie listy zawierającej wiele powtórzonych kopii

***`lista * n`***

```
In [11]: liczby = [0] * 5

In [12]: print(liczby)
[0, 0, 0, 0, 0]
```

Politechnika Wrocławska

# Listy i pętla `for`

- Wykorzystując pętle **`for`** można prowadzić iterację przez listę

```
In [13]: liczby = [99, 100, 101, 102]

In [14]: for n in liczby:
    ...:     print(n)
    ...:
99
100
101
102
```

Politechnika Wrocławska

# Listy i pętla `for`

- Taka lista może zawierać elementy różnych typów

```
In [15]: print(zawodnik)
['Bartosz Zmarzlik', 78, 2.41]

In [16]: for el in zawodnik:
    ...:     print(el)
    ...:
Bartosz Zmarzlik
78
2.41
```

Politechnika Wrocławska

# Indeksowanie

- Dostęp do poszczególnych elementów listy można uzyskać używając indeksów

```
In [17]: lista = [10, 20, 30, 40]

In [18]: lista[0]
Out[18]: 10

In [19]: lista[2]
Out[19]: 30

In [20]: print(lista[1], lista[3])
20 40
```

Politechnika Wrocławska

# Indeksowanie

- Użycie nieprawidłowego indeksu elementu powoduje zgłoszenie wyjątku

```
In [21]: print(lista)
[10, 20, 30, 40]

In [22]: lista[4]
Traceback (most recent call last):

  Cell In[22], line 1
    lista[4]

IndexError: list index out of range
```

Politechnika Wrocławska

# Indeksowanie

- Indeksy ujemne pozwalają na wskazywanie pozycji na liście od końca

```
In [23]: lista[-1]
Out[23]: 40

In [24]: lista[-4]
Out[24]: 10

In [25]: lista[-5]
Traceback (most recent call last):

  Cell In[25], line 1
    lista[-5]

IndexError: list index out of range
```

Politechnika Wrocławska

# Długość listy

- Do sprawdzenia długości listy można wykorzystać funkcję **`len()`**

```
In [27]: print(lista)
[10, 20, 30, 40]

In [28]: len(lista)
Out[28]: 4
```

Politechnika Wrocławska

# Długość listy

- Wykorzystując funkcję **`len()`** można przejść przez listę pętlą **`while`**

```
In [29]: print(lista)
[10, 20, 30, 40]

In [30]: index = 0

In [31]: while index < len(lista):
    ...:     print(lista[index])
    ...:     index += 1
    ...:
10
20
30
40
```

Politechnika Wrocławska

# Lista jest modyfikowalna

- Wartość elementów listy może być zmieniana

***lista[index] = nowa_wartosc***

```
In [32]: liczby = [1, 2, 3, 4, 5]

In [33]: print(liczby)
[1, 2, 3, 4, 5]

In [34]: liczby[2] = 77

In [35]: print(liczby)
[1, 2, 77, 4, 5]
```

Politechnika Wrocławska

# Lista jest modyfikowalna

- Takie przypisanie działa tylko dla pozycji, które już są na liście

```
In [36]: liczby[5] = 77
Traceback (most recent call last):

  Cell In[36], line 1
    liczby[5] = 77

IndexError: list assignment index out of range
```

Politechnika Wrocławska

# Lista jest modyfikowalna

01_tworzenie_listy.py

```
1  # Program pokazuje uzupełnienie danymi listy
2  # o ustalonej długości.
3
4  # Program prosi użytkownika o podanie
5  # temperatur powietrza odnotowanych przez
6  # 7 kolejnych dni.
7
8  #Stała NUM_DAYS określa liczbę dni, dla których
9  #będą pobierane dane od użytkownika.
```

Politechnika Wrocławska

# Lista jest modyfikowalna

01_tworzenie_listy.py

```
11  NUM_DAYS = 7
12
13  def main():
14      #utworzenie listy o ustalonej liczbie
15      #elementów - wszytskie równe zero
16      temperatures = [0]*NUM_DAYS
17
18      #pobranie od użytkownika temperatur z kolejnych dni
19      print('Podaj temperatury odnotowane przez 7 dni.')
20      index = 0
21      while index < NUM_DAYS:
22          print('Dzień nr ',index+1,': ',sep='',end='')
23          temperatures[index] = float(input())
24          index += 1
25
26      #wyświetlenie pobranych danych
27      print('Odnotowane temperatury')
28      for t in temperatures:
29          print(t)
30
31  main()
```

Politechnika Wrocławska

# Konkatencja list

- Do połączenia (konkatencji) list używa się operatora +

```
In [37]: lista1 = [1, 3, 5, 7]

In [38]: lista2 = [2, 4, 6, 8]

In [39]: lista3 = lista1 + lista2

In [40]: lista3
Out[40]: [1, 3, 5, 7, 2, 4, 6, 8]

In [41]: lista1
Out[41]: [1, 3, 5, 7]

In [42]: lista2
Out[42]: [2, 4, 6, 8]
```

Politechnika Wrocławska

# Konkatencja list

- Do połączenie list można użyć złożonego operatora przypisania +=

```
In [41]: lista1
Out[41]: [1, 3, 5, 7]

In [42]: lista2
Out[42]: [2, 4, 6, 8]

In [43]: lista2 += lista1

In [44]: lista2
Out[44]: [2, 4, 6, 8, 1, 3, 5, 7]
```

Politechnika Wrocławska

# Wycinek listy

- Z wykorzystaniem indeksu można uzyskać dostęp do pojedynczego elementu listy
- Python pozwala także tworzyć wycinek obejmujący pewien fragment listy

  ***nazwa_listy*[*początek*:*koniec*]**

Politechnika Wrocławska

# Wycinek listy

- Przykładowo:

```
In [45]: days_names = ['pn', 'wt', 'śr', 'cz', 'pt', 'sb', 'nd']

In [46]: days_names[0:5]
Out[46]: ['pn', 'wt', 'śr', 'cz', 'pt']

In [47]: days_names[2:5]
Out[47]: ['śr', 'cz', 'pt']
```

Politechnika Wrocławska

# Wycinek listy

- Pominięcie jednego z indeksów (początku lub końca) pozwala stworzyć wycinek listy od początku lub do końca:

```
In [48]: days_names[2:]
Out[48]: ['śr', 'cz', 'pt', 'sb', 'nd']

In [49]: days_names[:5]
Out[49]: ['pn', 'wt', 'śr', 'cz', 'pt']
```

Politechnika Wrocławska

# Wycinek listy

- Pominięcie obu indeksów pozwala stworzyć kopię całej listy:

```
In [50]: days_names[:]
Out[50]: ['pn', 'wt', 'śr', 'cz', 'pt', 'sb', 'nd']
```

Politechnika Wrocławska

# Wycinek listy

- Wycinek może zawierać również elementy listy z podanym krokiem:

```
In [51]: days_names[0:7:2]
Out[51]: ['pn', 'śr', 'pt', 'nd']

In [52]: days_names[0::2]
Out[52]: ['pn', 'śr', 'pt', 'nd']
```

Politechnika Wrocławska

# Wycinek listy

- Do wskazywania elementów wycinka można używać również ujemnych indeksów

```
In [53]: days_names[:-2]
Out[53]: ['pn', 'wt', 'śr', 'cz', 'pt']
```

Politechnika Wrocławska

# Wycinek listy

- Nieprawidłowy indeks nie spowoduje błędu:

```
In [54]: days_names[:10]
Out[54]: ['pn', 'wt', 'śr', 'cz', 'pt', 'sb', 'nd']

In [55]: days_names[-10:]
Out[55]: ['pn', 'wt', 'śr', 'cz', 'pt', 'sb', 'nd']

In [56]: days_names[5:2]
Out[56]: []
```

Politechnika Wrocławska

# Wyszukiwanie elementu listy

- Operator `in` pozwala sprawdzić, czy element znajduje się na liście
- Wywołanie:

```
element in lista
```

Politechnika Wrocławska

# Wyszukiwanie elementu listy

01_tworzenie_listy.py | 02_operator_in.py

```
1  # Program pokazuje sprawdzanie czy podany
2  # element znajduje się na liście.
3  # Program pokazuje użycie operatora in.
4
5  def main():
6      #utworzenie listy kolorów podstawowych
7      primary_colours = ['czerwony', 'zielony', 'niebieski']
8
9      #pobranie nazwy koloru od użytkownika
10     colour = input('Podaj nazwę koloru: ')
11
12     #sprawdzenie, czy podany kolor jest na liście
13     if colour in primary_colours:
14         print('Kolor',colour,'jest kolorem podstawowym.')
15     else:
16         print('Kolor',colour,'nie jest kolorem podstawowym.')
17
18
19 main()
```

Politechnika Wrocławska

# Wybrane metody listy

| metoda | opis |
|---|---|
| `append(`*`element`*`)` | dołącza element na końcu listy |
| `index(`*`element`*`)` | zwraca najniższy indeks, którego wartość odpowiada elementowi |
| `sort()` | porządkowanie elementów w kolejności rosnącej |
| `insert(`*`indeks, element`*`)` | wstawienie elementu na pozycji indeks |
| `remove(`*`element`*`)` | usuwa element z listy |
| `reverse()` | odwraca kolejność elementów na liście |

Politechnika Wrocławska

# Wybrane metody listy

## append()

01_tworzenie_listy.py | 02_operator_in.py | 03_append.py

```
1  # Program pokazuje użycie metody append()
2  # do dołączania elementów do listy.
3
4  # Program prosi użytkownika o wymienianie
5  # nazw kolorów jakie zna.
6
```

01_tworzenie_listy.py | 02_operator_in.py | 03_append.py

```
7  def main():
8      #utworzenie pustej listy
9      colours = []
10
11     #zmienna kontroluje przebieg pętli while
12     keep_going = 't'
13     while keep_going == 't':
14         #pobranie nazwy koloru od użytkownika
15         colour = input('Podaj kolor: ')
16
17         #dodanie nazwy do listy
18         colours.append(colour)
19
20         #pytanie czy kontynouwać wykonywanie pętli
21         print('Czy znasz jeszcze jakieś kolory?')
22         keep_going = input('Jeśli tak, wpisz t,'+
23                            ' w przeciwnym razie inny znak: ')
24
25     #wypisanie wszystkich kolorów
26     print('Wszystkie kolory jakie podałeś:')
27     for colour in colours:
28         print(colour)
29
30
31 main()
```

Politechnika Wrocławska

# Wybrane metody listy

## append()

01_tworzenie_listy.py | 02_operator_in.py | 03_append.py

```
1  # Program pokazuje użycie metody append()
2  # do dołączania elementów do listy.
3
4  # Program prosi użytkownika o wymienianie
5  # nazw kolorów jakie zna.
6
```

01_tworzenie_listy.py | 02_operator_in.py | 03_append.py

```
7  def main():
8      #utworzenie pustej listy
9      colours = []
10
11     #zmienna kontroluje przebieg pętli while
12     keep_going = 't'
13     while keep_going == 't':
14         #pobranie nazwy koloru od użytkownika
15         colour = input('Podaj kolor: ')
16
17         #dodanie nazwy do listy
18         colours.append(colour)
19
20         #pytanie czy kontynouwać wykonywanie pętli
21         print('Czy znasz jeszcze jakieś kolory?')
22         keep_going = input('Jeśli tak, wpisz t,'+
23                            ' w przeciwnym razie inny znak: ')
24
25     #wypisanie wszystkich kolorów
26     print('Wszystkie kolory jakie podałeś:')
27     for colour in colours:
28         print(colour)
29
30
31 main()
```

Politechnika Wrocławska

# Wybrane metody listy

## append()

01_tworzenie_listy.py | 02_operator_in.py | 03_append.py | 04_append_in.py

```
10      #zmienna kontroluje przebieg pętli while
11      keep_going = 't'
12      while keep_going == 't':
13          #pobranie nazwy koloru od użytkownika
14          colour = input('Podaj kolor: ')
15
16          #jeśli podanej nazwy nie ma liście
17          if colour not in colours:
18              #to ją dołącz
19              colours.append(colour)
20          else:
21              #w p. p. wyświetl komunikat i przerwij pętle
22              print('Kolor',colour,'jest już na liście.')
23              keep_going = 'n'
```

Politechnika Wrocławska

# Wybrane metody listy

## `index()`

```
11      #zmienna kontroluje przebieg pętli while
12      keep_going = 't'
13      while keep_going == 't':
14          #pobranie nazwy koloru od użytkownika
15          colour = input('Podaj kolor: ')
16
17          #jeśli podanej nazwy nie ma liście
18          if colour not in colours:
19              #to ją dołącz
20              colours.append(colour)
21          else:
22              #w p. p. wyświetl komunikat i przerwij pętle
23              print('Kolor ',colour,' jest już na liście. ',
24                    'Na pozycji ',colours.index(colour),'.',sep='')
25              keep_going = 'n'
26
```

Politechnika Wrocławska

# Wybrane metody listy

## `sort()`

06_sort.py

```
8   #zmienna kontroluje przebieg pętli while
9   keep_going = 't'
10  while keep_going == 't':
11      #pobranie nazwy koloru od użytkownika
12      colour = input('Podaj kolor: ')
13
14      #jeśli podanej nazwy nie ma liście
15      if colour not in colours:
16          #to ją dołącz
17          colours.append(colour)
18      else:
19          #w p. p. wyświetl komunikat i przerwij pętle
20          print('Kolor ',colour,' jest już na liście. ',
21                'Na pozycji ',colours.index(colour),'.',sep='')
22          keep_going = 'n'
23
24  #sortowanie listy w kolejności alfabetycznej
25  colours.sort()
26
27  #wypisanie wszystkich kolorów
28  print('Na liście są następujące kolory:')
29  for colour in colours:
30      print(colour)
```

Politechnika Wrocławska

# Wybrane metody listy

## `insert()`

```
In [58]: lista = [] # utworzenie pustej listy

In [59]: lista.insert(0, 'k') # wstawienie 'k' na pozycję 0

In [60]: lista
Out[60]: ['k']

In [61]: lista.insert(0,'g') # wstawienie 'g' na pozycję 0

In [62]: lista
Out[62]: ['g', 'k']

In [63]: # 'k' zostało przesunięte
```

Politechnika Wrocławska

# Wybrane metody listy

## `insert()`

```
In [58]: lista = [] # utworzenie pustej listy

In [59]: lista.insert(0, 'k') # wstawienie 'k' na pozycję 0

In [60]: lista
Out[60]: ['k']

In [61]: lista.insert(0,'g') # wstawie

In [62]: lista
Out[62]: ['g', 'k']

In [63]: # 'k' zostało przesunięte
```

```
In [64]: lista.insert(0,'a')

In [65]: lista
Out[65]: ['a', 'g', 'k']

In [66]: lista.insert(1,'b')

In [67]: lista
Out[67]: ['a', 'b', 'g', 'k']
```

Politechnika Wrocławska

# Wybrane metody listy

## `insert()`

- Użycie indeksu przekraczającego długość listy nie spowoduje błędu
- Element zostanie wstawiony na koniec lub początek listy

```
In [68]: lista.insert(10,'z')

In [69]: lista
Out[69]: ['a', 'b', 'g', 'k', 'z']

In [70]: lista.insert(-10,'z')

In [71]: lista
Out[71]: ['z', 'a', 'b', 'g', 'k', 'z']
```

Politechnika Wrocławska

# Wybrane metody listy

## `remove()`

- Metoda **`remove()`** usuwa z listy pierwsze wystąpienie elementu

```
In [71]: lista
Out[71]: ['z', 'a', 'b', 'g', 'k', 'z']

In [72]: lista.remove('z')

In [73]: lista
Out[73]: ['a', 'b', 'g', 'k', 'z']

In [74]: lista.remove('z')

In [75]: lista
Out[75]: ['a', 'b', 'g', 'k']

In [76]: lista.remove('z')
Traceback (most recent call last):

  Cell In[76], line 1
    lista.remove('z')

ValueError: list.remove(x): x not in list
```

Politechnika Wrocławska

# Wybrane metody listy

## `reverse()`

- Metoda **`reverse()`** odwraca porządek elementów na liście

```
In [77]: lista
Out[77]: ['a', 'b', 'g', 'k']

In [78]: lista.reverse()

In [79]: lista
Out[79]: ['k', 'g', 'b', 'a']
```

Politechnika Wrocławska

# Usuwanie elementu listy

## `del`

- Do usunięcia elementu listy można użyć polecenia **`del`**

```
In [80]: lista
Out[80]: ['k', 'g', 'b', 'a']

In [81]: del lista[1]

In [82]: lista
Out[82]: ['k', 'b', 'a']
```

Politechnika Wrocławska

# Znajdowanie minimum i maksimum

- Funkcje **`min()`** i **`max()`** zwracają wartość minimalnego i maksymalnego elementu z listy

```
In [82]: lista
Out[82]: ['k', 'b', 'a']

In [83]: min(lista)
Out[83]: 'a'

In [84]: max(lista)
Out[84]: 'k'
```

Politechnika Wrocławska

# Kopiowanie listy

- Przypisanie jednej zmiennej do drugiej powoduje odwoływanie się obu zmiennych do tego samego obszaru pamięci

```
In [89]: list1 = ['a', 'b', 'c', 'd']

In [90]: list2 = list1

In [91]: list1
Out[91]: ['a', 'b', 'c', 'd']

In [92]: list2
Out[92]: ['a', 'b', 'c', 'd']
```

Politechnika Wrocławska

# Kopiowanie listy

- Przypisanie jednej zmiennej do drugiej powoduje odwoływanie się obu zmiennych do tego samego obszaru pamięci

```
In [89]: list1 = ['a', 'b', 'c

In [90]: list2 = list1

In [91]: list1
Out[91]: ['a', 'b', 'c', 'd']

In [92]: list2
Out[92]: ['a', 'b', 'c', 'd']
```

Politechnika Wrocławska

# Kopiowanie listy

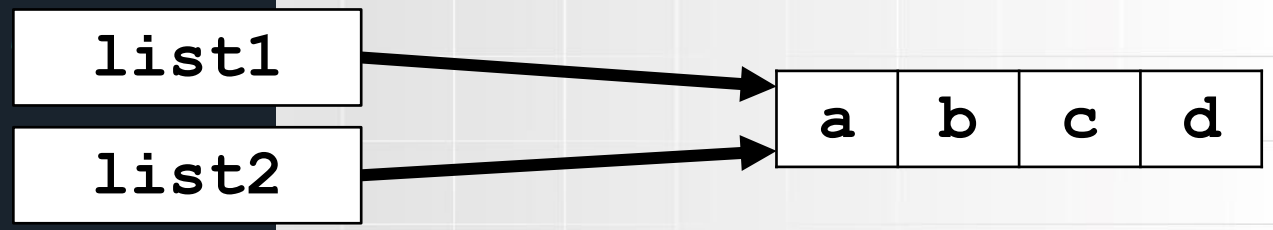
- Przypisanie jednej zmiennej do drugiej powoduje odwoływanie się obu zmiennych do tego samego obszaru pamięci

```
In [89]: list1 = ['a', 'b', 'c

In [90]: list2 = list1

In [91]: list1
Out[91]: ['a', 'b', 'c', 'd']

In [92]: list2
Out[92]: ['a', 'b', 'c', 'd']
```



Politechnika Wrocławska

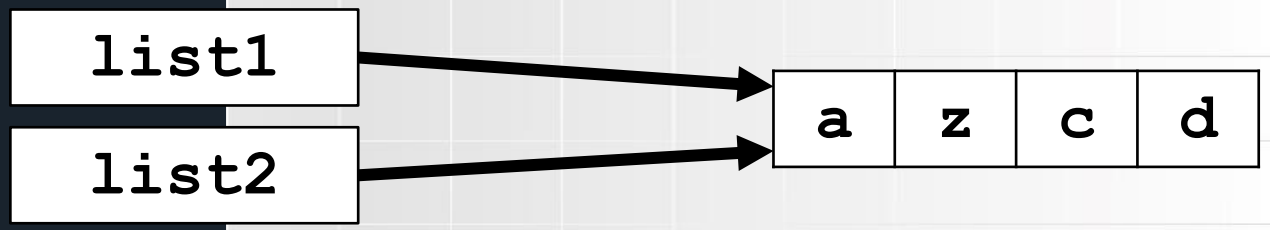
# Kopiowanie listy

- Obie zmienne wskazują na ten sam obszar pamięci. Zatem zmiana wartości elementu jest widoczna przez obie zmienne.

```
In [93]: list1[1] = 'z'

In [94]: list1
Out[94]: ['a', 'z', 'c', 'd']

In [95]: list2
Out[95]: ['a', 'z', 'c', 'd']
```



Politechnika Wrocławska

# Kopiowanie listy

- Do kopiowania listy można wykorzystać metodę `copy()`

```
In [103]: list1 = ['a', 'b', 'c', 'd']

In [104]: list2 = list1.copy()

In [105]: list1[1] = 'z'

In [106]: list1
Out[106]: ['a', 'z', 'c', 'd']

In [107]: list2
Out[107]: ['a', 'b', 'c', 'd']
```

Politechnika Wrocławska

# Przetwarzanie elementów listy

07_przetwarzanie_listy.py

```python
# Program pokazuje przetwarzanie listy.
# Program oblicza średnią liczb z listy.

def main():
    #lista z przykładowymi danymi
    data = [1,2,3,5,8,13]

    #obliczenie sumy elementów z listy
    total = 0
    for value in data:
        total += value
    #obliczenie średniej elementów
    average = total / len(data)
    #wypisanie wyniku
    print('Średnia danych z listy to',average)

main()
```

Politechnika Wrocławska

# Przetwarzanie elementów listy

07_przetwarzanie_listy.py | 08_przetwarzanie_listy.py

```python
# Program pokazuje przetwarzanie listy.
# Program oblicza średnią liczb z listy.

def main():
    #lista z przykładowymi danymi
    data = [1,2,3,5,8,13]
    #wywołanie funkcji average()
    #lista data przekazana jako argument
    print('Średnia danych z listy to',average(data))

#I: lista zawierająca dane do obliczeń
#P: funkcja oblicza sumę liczb na liście,
    # a następnie zwraca średnią
    # (iloraz sumy elementów i liczby elementów)
#O: średnia elementów listy
def average(data):
    total = 0
    for value in data:
        total += value
    return total / len(data)

main()
```

Politechnika Wrocławska

# Przetwarzanie elementów listy

09_przetwarzanie_listy.py

```python
# Program oblicza średnią liczb z listy.

def main():
    data = get_data(5)
    print('Średnia danych z listy to',average(data))

#funkcja zwraca średnią z elementów listy data
def average(data):
    total = 0
    for value in data:
        total += value
    avg = total / len(data)
    return avg

#I: liczba n reprezentująca długość listy
#P: wczytanie n liczb rzeczywistych i dodanie ich
    #do listy
#O: wczytana lista danych
def get_data(n):
    data = []
    for i in range(n):
        data.append(float(input('Podaj liczbę: ')))
    return data


main()
```

Politechnika Wrocławska

# Listy zagnieżdżone

```
In [120]: magiczny_kwadrat = [[6, 1, 8], [7, 5, 3], [2, 9, 4]]

In [121]: magiczny_kwadrat
Out[121]: [[6, 1, 8], [7, 5, 3], [2, 9, 4]]

In [122]: magiczny_kwadrat[0]
Out[122]: [6, 1, 8]

In [123]: magiczny_kwadrat[0][0]
Out[123]: 6

In [124]: magiczny_kwadrat[2][1]
Out[124]: 9
```

Politechnika Wrocławska

# Krotka

- Krotka to niemodyfikowalna sekwencja
- Do zapisania krotki używa się nawiasów okrągych

```
In [125]: tuple1 = (0, 1, 2)

In [126]: tuple1[1] = 2
Traceback (most recent call last):

  Cell In[126], line 1
    tuple1[1] = 2

TypeError: 'tuple' object does not support item assignment
```

Politechnika Wrocławska

# Krotka

- Krotka pozwala na odwoływanie się do elementów przez index
- Krotka może być argumentem funkcji **`len()`**, **`min()`**, **`max()`**
- Można tworzyć wycinki z krotki
- Krotka działa z operatorami **`in`**, **`+`**, **`*`**

Politechnika Wrocławska

# Krotka

- Funkcje **`list()`** i **`tuple()`** pozwalają przeprowadzić konwersję między krotką i listą

```
>>> krotka = (1,2,3)
>>> lista = list(krotka)
>>> print(lista)
[1, 2, 3]
>>> type(lista)
<class 'list'>
>>>
```

```
>>> lista = [4,5,6]
>>> krotka = tuple(lista)
>>> print(krotka)
(4, 5, 6)
>>> type(krotka)
<class 'tuple'>
>>>
```

Politechnika Wrocławska

# Krotka

- Funkcje **`list()`** i **`tuple()`** pozwalają przeprowadzić konwersję między krotką i listą

```
In [127]: krotka = (1, 2, 3)

In [128]: lista = list(krotka)

In [129]: lista
Out[129]: [1, 2, 3]

In [130]: type(lista)
Out[130]: list
```

```
In [131]: lista = [4, 5, 6]

In [132]: krotka = tuple(lista)

In [133]: krotka
Out[133]: (4, 5, 6)

In [134]: type(krotka)
Out[134]: tuple
```

Politechnika Wrocławska

# Rozpakowywanie krotki

- Krotki wykorzystywaliśmy do zwracania większej liczby wartości z funkcji
- Mechanizm ten bazował na rozpakowywaniu krotki

```
In [137]: k = (1, 2)

In [138]: a, b = k

In [139]: a
Out[139]: 1

In [140]: b
Out[140]: 2
```

Politechnika Wrocławska

# Rozpakowywanie krotki

12_rozpakowywanie_krotki.py

```python
# Program pokazuje wykorzystanie pakowania i rozpakowywania krotki
# do zwracania kilku wartości przez funkcję.

def main():
    # przypisanie dwóch wartości
    # zwracanych przez funkcję
    # do zmiennych x0, y0
    x0, y0 = fun(5, 8)
    print(x0, y0)

    # przypisanie dwóch wartości zapakowanych
    # w krotkę do zmiennej k
    k = fun(5, 8)
    print(type(k)) # <class 'tuple'>
    print(k)       # wyświetlenie krotki

    x1, y1 = k     # rozpakowanie krotki do zmiennych x1, y1
    print(x1, y1)
```

Politechnika Wrocławska

# Rozpakowywanie krotki

```
21  # funkcja przyjmuje dwa argumenty
22  # oblicza ich sumę oraz różnicę
23  # wartości zwraca w dwuelementowej krotce
24  def fun(a, b):
25      s = a + b
26      r = a - b
27      return (s, r)
28
29  main()
```

Politechnika Wrocławska

# Rozpakowywanie listy/krotki

- Rozpakowywanie pozwala przypisać elementy krotki/listy do kilku zmiennych

```
In [137]: k = (1, 2)

In [138]: a, b = k

In [139]: a
Out[139]: 1

In [140]: b
Out[140]: 2
```

Politechnika Wrocławska

# Rozpakowywanie listy/krotki

- Liczba zmiennych po lewej stronie operatora przypisania powinna odpowiadać liczbie elementów krotki/listy

```
In [141]: t = tuple(range(5))

In [142]: t
Out[142]: (0, 1, 2, 3, 4)

In [143]: a, b = t
Traceback (most recent call last):

  Cell In[143], line 1
    a, b = t

ValueError: too many values to unpack (expected 2)
```

Politechnika Wrocławska

# Rozpakowywanie listy/krotki

- Jeśli nie znamy liczby argumentów możemy wykorzystać operator *

```
In [144]: t = tuple(range(5))

In [145]: t
Out[145]: (0, 1, 2, 3, 4)

In [146]: a, *b = t

In [147]: a
Out[147]: 0

In [148]: b
Out[148]: [1, 2, 3, 4]

In [149]: type(a)
Out[149]: int

In [150]: type(b)
Out[150]: list
```

Politechnika Wrocławska

# Rozpakowywanie listy/krotki

- Jeśli nie znamy liczby argumentów możemy wykorzystać operator *

```
In [151]: t
Out[151]: (0, 1, 2, 3, 4)

In [152]: first, *middle, last = t

In [153]: first
Out[153]: 0

In [154]: middle
Out[154]: [1, 2, 3]

In [155]: last
Out[155]: 4
```

Politechnika Wrocławska

# Operator *

- Operator * (pakowania/rozpakowania) można wykorzystać do przekształcenia listy w kilka argumentów pozycyjnych

```
In [1]: list(range(2,5))
Out[1]: [2, 3, 4]

In [2]: list(range([2,5]))
Traceback (most recent call last):

  Cell In[2], line 1
    list(range([2,5]))

TypeError: 'list' object cannot be interpreted as an integer


In [3]: list(range(*[2,5]))
Out[3]: [2, 3, 4]
```

# Operator *

- Operator * można wykorzystać do zapakowania kilku argumentów pozycyjnych w krotkę

Politechnika Wrocławska

# Operator *

variable_argument_list_07.py

```
1  # Program pokazuje przekazywanie argumentów pozycyjnych.
2  # Liczba argumentów w wywołaniu musi wynosić 3.
3  def fun(a,b,c):
4      print(a)
5      print(b)
6      print(c)
7
8  #fun('a', 'b')
9  fun('a', 'b', 'c')
10 #fun('a', 'b', 'c', 'd')
11
```

Politechnika Wrocławska

# Operator *

variable_argument_list_07.py | variable_argument_list_09.py

```python
# Program pokazuje funkcję przyjmującą wiele argumentów pozycyjnych.
def fun(*args):
    for item in args:
        print(item)

fun('a', 'b', 'c')
fun('a', 'b', 'c', 'd')
fun('a')
fun()
```

Politechnika Wrocławska

# Operator *

variable_argument_list_07.py | variable_argument_list_09.py

```python
# Program pokazuje funkcję przyjmującą wiele argumentów pozycyjnych.
def fun(*args):
    for item in args:
        print(item)

fun('a', 'b', 'c')
fun('a', 'b', 'c', 'd')
fun('a')
fun()
```

Politechnika Wrocławska

# Operator *

variable_argument_list_07.py | variable_argument_list_21.py

```
1  # Program pokazuje funkcję przyjmującą co najmniej 2 argumenty pozycyjne.
2  def fun(a, b, *args):
3      print(a)
4      print(b)
5      print(args)
6
7  fun('a', 'b', 'c')
8  fun('a', 'b', 'c', 'd')
9  #fun('a')
10 #fun()
11
```

Politechnika Wrocławska

# Podsumowanie (1)

- Operator powtórzenia `*`
- Listy i pętla `for`
- Indeksowanie elementów `[]`
- Długość listy `len()`

Politechnika Wrocławska

# Podsumowanie (2)

- Modyfikowanie list `lista[] =`
- Konkatenacja list `+`
- Wycinek listy `[a:b]`
- Wyszukiwanie elementu listy `in`
- Wybrane metody listy
- Kopiowanie listy `copy()`
- Przetwarzanie danych
- Listy zagnieżdżone `lista = [[1, 2], [3, 4]]`

Politechnika Wrocławska

# Podsumowanie (3)

- Krotki `tuple()  (1,2,3)`
- Rozpakowywanie krotek `a,b = (1,2)`

- Zmienna liczba argumentów funkcji `def fun(*args):`