

Podstawy programowania W11OPA-SI0072G

Wstęp do programowania W11FTE-SI0141WL

Wstęp do programowania W11IKW-SI0080WL

rok akademicki 2024/25

semestr letni

Wykład 4

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 221

Plan prezentacji (1)

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

Plan prezentacji (2)

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

Plan prezentacji (3)

- Krotki (niemodyfikowalne sekwencje)
- Rozpakowywanie krotek
- Zmienna liczba argumentów wywołania funkcji

Wprowadzenie do list

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

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]
```

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']
```

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]
```


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]
```

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]
```

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
```

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
```

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
```

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
```

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
```

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
```


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
```

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]
```

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
```

Lista jest modyfikowalna

```
01_tworzenie_listy.py X
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.
10
```

Lista jest modyfikowalna

```
01_tworzenie_listy.py X
11 NUM_DAYS = 7
12
13 def main():
14     #utworzenie listy o ustalonej liczbie
15     #elementów - wszystkie 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()
```

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]
```

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]
```

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]`

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']
```

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']
```

Wycinek listy

- Pominięcie obu indeksów pozwala stworzyć kopię całej listy:

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

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']
```

Wycinek listy

- Do wskazywania elementów wycinka można używać również ujemnych indeksów

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

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]: []
```

Wyszukiwanie elementu listy

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

Wyszukiwanie elementu listy

```
01_tworzenie_listy.py X 02_operator_in.py X
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()
20
```


Wybrane metody listy

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

Wybrane metody listy

append()

```
01_tworzenie_listy.py X 02_operator_in.py X 03_append.py X
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 X 02_operator_in.py X 03_append.py X
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 kontynuować 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()
32
```

Wybrane metody listy

append()

```
01_tworzenie_listy.py X 02_operator_in.py X 03_append.py X
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 X 02_operator_in.py X 03_append.py X
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 kontynuować 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()
32
```

Wybrane metody listy

append()

```
01_tworzenie_listy.py X 02_operator_in.py X 03_append.py X 04_append_in.py X
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 pętlę
22             print('Kolor', colour, 'jest już na liście.')
23             keep_going = 'n'
```

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
```

Wybrane metody listy

sort()

```
06_sort.py X
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)
```

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
```

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
```

```
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']
```


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']
```

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
```

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']
```

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']
```

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'
```

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']
```

Kopiowanie listy

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

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

list1

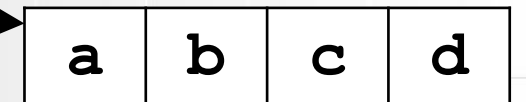
```
In [90]: list2 = list1
```

```
In [91]: list1
```

```
Out[91]: ['a', 'b', 'c', 'd']
```

```
In [92]: list2
```

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



Kopiowanie listy

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

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

list1

```
In [90]: list2 = list1
```

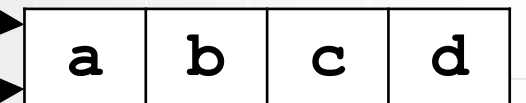
list2

```
In [91]: list1
```

```
Out[91]: ['a', 'b', 'c', 'd']
```

```
In [92]: list2
```

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



Kopiowanie listy

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

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

list1

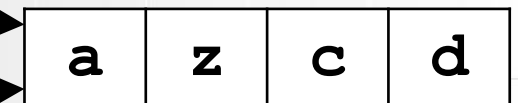
```
In [94]: list1
```

```
Out[94]: ['a', 'z', 'c', 'd']
```

```
In [95]: list2
```

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

list2



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']
```

Przetwarzanie elementów listy



07_przetwarzanie_listy.py X

```
1  # Program pokazuje przetwarzanie listy.
2  # Program oblicza średnią liczb z listy.
3
4  def main():
5      #lista z przykładowymi danymi
6      data = [1,2,3,5,8,13]
7
8      #obliczenie sumy elementów z listy
9      total = 0
10     for value in data:
11         total += value
12     #obliczenie średniej elementów
13     average = total / len(data)
14     #wypisanie wyniku
15     print('Średnia danych z listy to',average)
16
17 main()
```

Przetwarzanie elementów listy

```
07_przetwarzanie_listy.py X 08_przetwarzanie_listy.py X
1  # Program pokazuje przetwarzanie listy.
2  # Program oblicza średnią liczb z listy.
3
4  def main():
5      #lista z przykładowymi danymi
6      data = [1,2,3,5,8,13]
7      #wywołanie funkcji average()
8      #lista data przekazana jako argument
9      print('Średnia danych z listy to',average(data))
10
11  #I: lista zawierająca dane do obliczeń
12  #P: funkcja oblicza sumę liczb na liście,
13     # a następnie zwraca średnią
14     # (iloraz sumy elementów i liczby elementów)
15  #O: średnia elementów listy
16  def average(data):
17      total = 0
18      for value in data:
19          total += value
20      return total / len(data)
21
22  main()
```

Przetwarzanie elementów listy

```
09_przetwarzanie_listy.py X
2  # Program oblicza średnią liczb z listy.
3
4  def main():
5      data = get_data(5)
6      print('Średnia danych z listy to', average(data))
7
8  #funkcja zwraca średnią z elementów listy data
9  def average(data):
10     total = 0
11     for value in data:
12         total += value
13     avg = total / len(data)
14     return avg
15
16     #I: liczba n reprezentująca długość listy
17     #P: wczytanie n liczb rzeczywistych i dodanie ich
18     #do listy
19     #O: wczytana lista danych
20     def get_data(n):
21         data = []
22         for i in range(n):
23             data.append(float(input('Podaj liczbę: ')))
24         return data
25
26
27     main()
```

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
```

Krotka

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

```
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
```

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`, `+`, `*`

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
```

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
```

Rozpakowywanie krotki

```
12_rozpakowywanie_krotki.py X
1  # Program pokazuje wykorzystanie pakowania i rozpakowywania krotki
2  # do zwracania kilku wartości przez funkcję.
3
4  def main():
5      # przypisanie dwóch wartości
6      # zwracanych przez funkcję
7      # do zmiennych x0, y0
8      x0, y0 = fun(5, 8)
9      print(x0, y0)
10
11     # przypisanie dwóch wartości zapakowanych
12     # w krotkę do zmiennej k
13     k = fun(5, 8)
14     print(type(k)) # <class 'tuple'>
15     print(k)       # wyświetlenie krotki
16
17     x1, y1 = k      # rozpakowanie krotki do zmiennych x1, y1
18     print(x1, y1)
19
```

Rozpakowywanie krotki

```
20
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()
30
```

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
```

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)
```

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
```

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
```


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

Operator *

```
variable_argument_list_07.py X
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
```

Operator *

```
variable_argument_list_07.py X variable_argument_list_09.py X
1  # Program pokazuje funkcję przyjmującą wiele argumentów pozycyjnych.
2  def fun(*args):
3      for item in args:
4          print(item)
5
6  fun('a', 'b', 'c')
7  fun('a', 'b', 'c', 'd')
8  fun('a')
9  fun()
10
```

Operator *

```
variable_argument_list_07.py X variable_argument_list_21.py X
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
```

Podsumowanie (1)

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

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]]`

Podsumowanie (3)

- Krotki `tuple()` `(1,2,3)`
- Rozpakowywanie krotek `a,b = (1,2)`
- Zmienna liczba argumentów funkcji `def fun(*args) :`