

Podstawy programowania W11OPA-SI0072G

Wstęp do programowania W11FTE-SI0141WL

Wstęp do programowania W11IKW-SI0080WL

rok akademicki 2024/25
semestr letni

Wykład 5

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 221

Plan prezentacji (1)

- Słowniki
 - tworzenie słownika
 - operacje na słowniku - wybrane operatory, funkcje, metody
- Przekazywanie do funkcji zmiennej liczby argumentów nazwanych
- Zbiory
 - tworzenie zbiorów
 - operacje na zbiorach - wybrane operatory, funkcje, metody

Plan prezentacji (2)

- Znaki jako liczby
- Dostęp do znaków ciągu tekstowego
- Ciąg tekstowy jest niemodyfikowalny
- Wycinki ciągu tekstowego
- Wybrane metody ciągu tekstowego

Słownik

- Słownik to obiekt przechowujący dane w postaci par: klucz-wartość.

Utworzenie słownika

- Przykład polecenia tworzącego słownik:

```
dictionary = {1: 'pn', 2: 'wt', 3: 'śr'}
```

```
In [1]: dictionary = {1 : 'pn', 2 : 'wt', 3 : 'śr'}
```

```
In [2]: print(dictionary)  
{1: 'pn', 2: 'wt', 3: 'śr'}
```

Różne typy wartości

- Wartości przechowywane w słowniku mogą być różnych typów

```
In [3]: movie = {'title': 'The Shawshank Redemption', 'year': 1994, 'grade' : 8.8}
```

```
In [4]: print(movie)
```

```
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.8}
```

Różne typy kluczy

- Klucze również mogą być różnych typów

```
In [5]: divisors = {1: [1], 'two': [1, 2], '6': [1, 2, 3, 6] }
```

```
In [6]: print(divisors)  
{1: [1], 'two': [1, 2], '6': [1, 2, 3, 6]}
```

Pobieranie wartości ze słownika

- Elementy słownika nie są przechowywane w ustalonej kolejności
- Nie ma możliwości dostępu do wartości na podstawie pozycji
- Do pobrania wartości używa się klucza
`dictionary[key]`

```
In [7]: dictionary[2]
```

```
Out[7]: 'wt'
```

```
In [8]: movie['title']
```

```
Out[8]: 'The Shawshank Redemption'
```

```
In [9]: divisors['6']
```

```
Out[9]: [1, 2, 3, 6]
```

Pobieranie wartości ze słownika

- Użycie złego klucza powoduje błąd `KeyError`

```
In [10]: dictionary[4]  
Traceback (most recent call last):
```

```
Cell In[10], line 1  
    dictionary[4]
```

```
KeyError: 4
```

```
In [11]: divisors['six']  
Traceback (most recent call last):
```

```
Cell In[11], line 1  
    divisors['six']
```

```
KeyError: 'six'
```

Pobieranie wartości ze słownika

- Wielkość liter ma znaczenie

```
In [12]: movie['Title']  
Traceback (most recent call last):  
  
  Cell In[12], line 1  
        movie['Title']  
  
KeyError: 'Title'
```

Sprawdzanie wartości w słowniku

- Aby uniknąć błędu można sprawdzić, czy klucz istnieje w słowniku
- Do sprawdzenia można wykorzystać operator `in`

```
In [14]: if 'Title' in movie:
...:     print(movie['Title'])
...:
```

```
In [15]: if 'title' in movie:
...:     print(movie['title'])
...:
```

The Shawshank Redemption

Sprawdzanie wartości w słowniku

- Można również użyć operatora `not in`

```
In [16]: if 3 not in divisors:
...:     print('3 nie jest kluczem.')
...:
3 nie jest kluczem.
```

Dodanie elementu do słownika

- Słownik jest obiektem modyfikowalnym
- Polecenie

`dictionary[key] = value`

- zmodyfikuje wartość dla klucza **`key`** jeśli ten klucz istnieje w słowniku
- doda nową parę do słownika w przeciwnym przypadku

Dodanie elementu do słownika

- Słownik jest
- Polecenie `dictionary`
 - zmodyfikuj
 - doda now

```
In [17]: movie
Out[17]: {'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.8}

In [18]: movie['grade'] = 8.7

In [19]: movie
Out[19]: {'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7}

In [20]: movie['country'] = 'United States'

In [21]: movie
Out[21]: {'title': 'The Shawshank Redemption',
'year': 1994,
'grade': 8.7,
'country': 'United States'}
```

Usunięcie elementu ze słownika

- Polecenie `del` pozwala usunąć parę klucz-wartość `del dictionary[key]`

```
In [22]: del movie['country']
```

```
In [23]: movie
```

```
Out[23]: {'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7}
```

Usunięcie elementu ze słownika

- Jeśli klucz nie istnieje, to pojawi się błąd **KeyError**

```
In [24]: del movie['garde']  
Traceback (most recent call last):  
  
  Cell In[24], line 1  
        del movie['garde']  
  
KeyError: 'garde'
```

Sprawdzenie liczby elementów

- Do sprawdzenia liczby elementów można wykorzystać funkcję `len()`

```
In [26]: movie
Out[26]: {'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7}

In [27]: len(movie)
Out[27]: 3
```

Klucze muszą być niemodyfikowalne

- Obiekt modyfikowalny nie może być kluczem

```
In [28]: mixed = {} #utworzenie nowego słownika
In [29]: mixed['abc'] = 1 #dodanie pary klucz wartość
In [30]: #klucz jest napisem, wartość liczbą
In [31]: mixed[4] = 'def' #klucz jest liczba, wartość napisem
In [32]: mixed[(2,3,5,7)] = [2,3,5,7] #klucz jest krotką, wartość listą

In [33]: mixed
Out[33]: {'abc': 1, 4: 'def', (2, 3, 5, 7): [2, 3, 5, 7]}

In [34]: mixed[[11,13]] = 'to się nie uda' #klucz nie może być listą
Traceback (most recent call last):

  Cell In[34], line 1
    mixed[[11,13]] = 'to się nie uda' #klucz nie może być listą

TypeError: unhashable type: 'list'
```

Tworzenie słownika funkcją `dict()`

- Funkcja `dict()` tworzy słownik
- Funkcja `dict()` pozwala tworzyć słownik przez argumenty nazwane - nazwa argumentu staje się kluczem, a wartość argumentu staje się wartością w parze
- Funkcja `dict()` pozwala tworzyć słownik z sekwencji par klucz-wartość

Tworzenie słownika funkcją dict()

```
In [35]: a = dict(one=1, two=2, three=3)
```

```
In [36]: a
```

```
Out[36]: {'one': 1, 'two': 2, 'three': 3}
```

```
In [37]: b = {'one':1, 'two':2, 'three': 3 }
```

```
In [38]: b
```

```
Out[38]: {'one': 1, 'two': 2, 'three': 3}
```

```
In [39]: c=dict([('two',2),('one',1), ('three',3)])
```

```
In [40]: c
```

```
Out[40]: {'two': 2, 'one': 1, 'three': 3}
```

```
In [41]: d = dict({'three':3,'one':1,'two':2})
```

```
In [42]: d
```

```
Out[42]: {'three': 3, 'one': 1, 'two': 2}
```

```
In [43]: a == b == c == d
```

```
Out[43]: True
```

Słowniki i pętla `for`

- Wykorzystując pętle `for` można prowadzić iterację przez wszystkie klucze słownika

```
In [50]: for key in movie:
...:     print(key)
...:
title
year
grade
```

Słowniki i pętla for

- Dla każdego klucza można sprawdzić wartość,

```
In [51]: for key in movie:
...:     print(key, movie[key])
...:
title The Shawshank Redemption
year 1994
grade 8.7
```

nie jest to dobra metoda...

Słowniki i pętla `for`

- Ten sam efekt można osiągnąć wykorzystując metodę `items()`, która zwraca widok słownika

```
In [52]: for key, value in movie.items():  
        ...:     print(key, value)  
        ...:  
title The Shawshank Redemption  
year 1994  
grade 8.7
```

Metody słownika

- Metoda `items()` zwraca widok słownika (sekwencję zawierającą dwuelementowe krotki przechowujące pary klucz-wartość)
- Metoda `values()` zwraca widok słownika zawierający wszystkie wartości słownika
- Metoda `keys()` zwraca widok słownika zawierający wszystkie klucze słownika

Metody słownika

- Metoda `items()` zwraca widok słownika (sekwencję zawierającą dwuelementowe krotki przechowujące pary klucz-wartość)
- Metoda `values()` zwraca widok słownika wszystkie wartości słownika

```
In [54]: movie.items()
```

```
Out[54]: dict_items([('title', 'The Shawshank Redemption'), ('year', 1994), ('grade', 8.7)])
```

```
In [55]: movie.values()
```

```
Out[55]: dict_values(['The Shawshank Redemption', 1994, 8.7])
```

```
In [56]: movie.keys()
```

```
Out[56]: dict_keys(['title', 'year', 'grade'])
```

Metody słownika

- Metoda `get()` pozwala odczytać wartość ze słownika
- Jeśli klucza nie ma w słowniku zwraca wartość domyślną
`dictionary.get(key, default)`

```
In [57]: movie.get('title', 'unknown key')
```

```
Out[57]: 'The Shawshank Redemption'
```

```
In [58]: movie.get('country', 'unknown key')
```

```
Out[58]: 'unknown key'
```

Metody słownika

- Metoda `pop()` działa podobnie do metody `get()`
- Dodatkowo usuwa parę klucz-wartość ze słownika

```
In [59]: movie.pop('country', 'unknown key')
```

```
Out[59]: 'unknown key'
```

```
In [60]: movie.pop('title', 'unknown key')
```

```
Out[60]: 'The Shawshank Redemption'
```

```
In [61]: movie
```

```
Out[61]: {'year': 1994, 'grade': 8.7}
```

Metoda `popitem()`

- Metoda `popitem()` działa podobnie do metody `pop()`
- Zwraca jedną parę klucz-wartość (LIFO) i usuwa wpis ze słownika

```
In [62]: k, v = movie.popitem()
```

```
In [63]: print(k, v)  
grade 8.7
```

```
In [64]: print(movie)  
{ 'year': 1994 }
```

Metoda `clear()`

- Metoda `clear()` usuwa zawartość słownika

```
In [65]: movie.clear()
```

```
In [66]: movie
```

```
Out[66]: {}
```

Wykorzystanie słownika

Przekazywanie argumentów do funkcji

- Przypomnienie
- Funkcja przyjmująca stałą liczbę argumentów

```
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
```

Wykorzystanie słownika

Przekazywanie argumentów do funkcji

- Przypomnienie
- Przekazywanie argumentów nazwanych

```
variable_argument_list_07.py X variable_argument_list_08.py X
1  # Program pokazuje przekazywanie argumentów nazwanych.
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 = 'a', b = 'b', c = 'c')
9  fun(b = 'b', c = 'c', a = 'a')
10
```

Wykorzystanie słownika

Przekazywanie argumentów do funkcji

- Przypomnienie
- Lista argumentów o zmiennej długości

```
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
```

Wykorzystanie słownika

Przekazywanie argumentów do funkcji

- Lista argumentów o zmiennej długości z argumentami nazwanymi - błąd!

```
variable_argument_list_09.py X variable_argument_list_10.py X
1  # Program pokazuje wywołanie funkcji
2  # przyjmującej wiele argumentów pozycyjnych
3  # z argumentami nazwanymi.
4  def fun(*args):
5      for item in args:
6          print(item)
7
8  fun(a = '1', b = '2', c = '3')
```

Wykorzystanie słownika

Przekazywanie argumentów do funkcji

- Lista argumentów o zmiennej długości z argumentami nazwanymi
 - rozpakowanie słownika

```
variable_argument_list_11.py X
1  # Program pokazuje funkcję przyjmującą wiele argumentów nazwanych.
2  def fun(**kwargs):
3      for key, item in kwargs.items():
4          print(key,item,sep=':')
5
6  fun(a = '1', b = '2', c = '3')
7
```

Wykorzystanie słownika

Przekazywanie argumentów do funkcji

- Lista argumentów o zmiennej długości z argumentami pozycyjnymi oraz nazwanymi

```
variable_argument_list_12.py X
1  # Program pokazuje funkcję przyjmującą:
2  # wymagane argumenty (a, b, c),
3  # różną liczbę argumentów pozycyjnych
4  # różną liczbę argumentów nazwanych.
5  def fun(a, b, c, *args, **kwargs):
6      print(a)
7      print(b)
8      print(c)
9
10     for item in args:
11         print(item)
12
13     for key, value in kwargs.items():
14         print(key, value, sep=':')
15
16 fun('1', '2', '3', '4', '5', k1 = 'v1', k2 = 'v2', k3 = 'v3')
17 # fun('1', '2')
18
```

Zbiór

- Zbiór to obiekt przechowujący kolekcję unikatowych wartości

Funkcja `set()`

- Do utworzenia zbioru służy funkcja `set()`
- Funkcja `set()` może być wywołana bez argumentów
- Funkcja `set()` może być wywołana z argumentem w postaci listy
- Funkcja `set()` może być wywołana z argumentem w postaci łańcucha znakowego

Funkcja `set()`

```
In [77]: empty_set = set()
```

```
In [78]: set_abc = set(['a', 'b', 'c'])
```

```
In [79]: char_set = set('abc')
```

```
In [80]: empty_set
```

```
Out[80]: set()
```

```
In [81]: set_abc
```

```
Out[81]: {'a', 'b', 'c'}
```

```
In [82]: char_set
```

```
Out[82]: {'a', 'b', 'c'}
```

Funkcja `set()`

- Zbiór zawiera tylko unikatowe elementy

```
In [83]: set('WPPT PWr')  
Out[83]: {' ', 'P', 'T', 'W', 'r'}
```

Funkcja `set()`

- Funkcja `set()` pobiera maksymalnie 1 argument

```
In [84]: set('WPPT', 'PWr')
Traceback (most recent call last):

  Cell In[84], line 1
        set('WPPT', 'PWr')

TypeError: set expected at most 1 argument, got 2
```

- Przykład tworzenia zbioru zawierającego łańcuchy znakowe

```
In [85]: set(['WPPT', 'PWr'])
Out[85]: {'PWr', 'WPPT'}
```

Sprawdzenie liczby elementów

- Do sprawdzenia liczby elementów można wykorzystać funkcję `len()`

```
In [86]: chars = set('WPPT PWr')
```

```
In [87]: chars
```

```
Out[87]: {' ', 'P', 'T', 'W', 'r'}
```

```
In [88]: len(chars)
```

```
Out[88]: 5
```

Dodawanie elementów do zbioru

- Dodawanie elementów metodą `add()`

```
In [1]: myset = set()
```

```
In [2]: myset.add(1)
```

```
In [3]: myset.add(2)
```

```
In [4]: myset.add(3)
```

```
In [5]: myset
```

```
Out[5]: {1, 2, 3}
```

```
In [6]: myset.add(2)
```

```
In [7]: myset
```

```
Out[7]: {1, 2, 3}
```

Dodawanie elementów do zbioru

- Dodawanie elementów metodą `update()`

```
In [8]: myset.update([4, 5, 6])
```

```
In [9]: myset
```

```
Out[9]: {1, 2, 3, 4, 5, 6}
```

Dodawanie elementów do zbioru

- Dodawanie elementów metodą `update()`

```
In [10]: myset2 = set([7, 8, 9])
```

```
In [11]: myset.update(myset2)
```

```
In [12]: myset
```

```
Out[12]: {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Usuwanie elementów ze zbioru

- Usuwanie elementów metodą `remove()`

```
In [13]: myset.remove(2)
```

```
In [14]: myset
```

```
Out[14]: {1, 3, 4, 5, 6, 7, 8, 9}
```

```
In [15]: myset.remove(2)
```

```
Traceback (most recent call last):
```

```
Cell In[15], line 1
```

```
    myset.remove(2)
```

```
KeyError: 2
```

Usuwanie elementów ze zbioru

- Usuwanie elementów metodą `discard()`

```
In [16]: myset.discard(3)

In [17]: myset
Out[17]: {1, 4, 5, 6, 7, 8, 9}

In [18]: myset.discard(3)

In [19]: myset
Out[19]: {1, 4, 5, 6, 7, 8, 9}
```

- Metoda `discard()` nie zgłasza błędu, gdy elementu nie ma w zbiorze

Usuwanie wszystkich elementów zbioru

- Czyszczenie zbioru metodą `clear()`

```
In [20]: myset.clear()
```

```
In [21]: myset
```

```
Out[21]: set()
```

Przechodzenie przez zbiór

- Do przejścia po elementach zbioru możemy wykorzystać pętlę **for**

```
In [22]: myset = set([-2, -1, 0, 1, 2])
```

```
In [23]: for el in myset:  
...:     print(el)  
...:
```

```
0  
1  
2  
-1  
-2
```

Sprawdzanie przynależności do zbioru

- Na zbiorach działają operatory `in` oraz `not in`

```
In [24]: myset  
Out[24]: {-2, -1, 0, 1, 2}
```

```
In [25]: 1 in myset  
Out[25]: True
```

```
In [26]: -5 in myset  
Out[26]: False
```

```
In [27]: 1 not in myset  
Out[27]: False
```

```
In [28]: -5 not in myset  
Out[28]: True
```

Łączenie zbiorów

- Do łączenia zbiorów można wykorzystać metodę `union()` lub operator `|`

```
In [33]: set4 = set1
```

```
In [34]: set1 = {1,2,3,4}
```

```
In [35]: set2 = {3,4,5,6}
```

```
In [36]: set3 = set1.union(set2)
```

```
In [37]: set3
```

```
Out[37]: {1, 2, 3, 4, 5, 6}
```

```
In [38]: set4 = set1 | set2
```

```
In [39]: set4
```

```
Out[39]: {1, 2, 3, 4, 5, 6}
```

Część wspólna zbiorów

- Do wyznaczenia części wspólnej można wykorzystać metodę `intersection()` lub operator `&`

```
In [40]: set5 = set1.intersection(set2)
```

```
In [41]: set5
```

```
Out[41]: {3, 4}
```

```
In [42]: set6 = set1 & set2
```

```
In [43]: set6
```

```
Out[43]: {3, 4}
```

Różnica zbiorów

- Do wyznaczenia różnicy zbiorów można wykorzystać metodę `difference()` lub operator –

```
In [44]: set7 = set1.difference(set2)
```

```
In [45]: set7
```

```
Out[45]: {1, 2}
```

```
In [46]: set8 = set2 - set1
```

```
In [47]: set8
```

```
Out[47]: {5, 6}
```

Różnica symetryczna zbiorów

- Do wyznaczenia różnicy symetrycznej zbiorów można wykorzystać metodę `symmetric_difference()` lub operator `^`

```
In [64]: set9 = set1.symmetric_difference(set2)
```

```
In [65]: set10 = set1 ^ set2
```

```
In [66]: set9
```

```
Out[66]: {1, 2, 5, 6}
```

```
In [67]: set10
```

```
Out[67]: {1, 2, 5, 6}
```

Wyszukiwanie podzbioru

- Do sprawdzenia zawierania się zbiorów można wykorzystać metody `issubset()` oraz `issuperset()` lub operatory `<=` oraz `>=`

```
In [52]: set1 = set('abcdef')  
  
In [53]: set2 = set('ae')  
  
In [54]: set2.issubset(set1)  
Out[54]: True  
  
In [55]: set1.issuperset(set2)  
Out[55]: True  
  
In [56]: set2 <= set1  
Out[56]: True  
  
In [57]: set1 >= set2  
Out[57]: True
```

Znaki jako liczby

Symbole znakowe są przechowywane w pamięci komputera jako liczby.

Najbardziej podstawowy system kodowania to system ASCII (American Standard Code for Information Interchange)

- literom od „A” do „Z” odpowiadają liczby od 65 do 90,
- literom od „a” do „z” odpowiadają liczby od 97 do 122,
- cyfrom od „0” do „9” odpowiadają liczby od 48 do 57,
- znakowi spacji odpowiada liczba 32,
- znakowi nowej linii odpowiada liczba 10.

Kody ASCII nie obejmują polskich znaków diakrytycznych.

Znaki jako liczby

- Rozszerzeniem zestawu znaków ASCII na znaki używane w różnych językach jest zestaw znaków Unicode
- Do zapisywania znaków Unicode wykorzystuje się różne kodowania
- Popularnym kodowaniem jest kodowanie UTF-8 (Unicode Transformation Format), gdzie 8 oznacza, że do kodowania wykorzystywane są liczby 8-bitowe

Znaki jako liczby

- Funkcja `ord()` dla podanego znaku zwraca jego kod liczbowy
- Funkcja `chr()` dla podanego kodu liczbowego zwraca odpowiadający mu znak

Porównywanie ciągów znakowych

- Porównywanie ciągów tekstowych sprowadza się do porównywania kodów kolejnych znaków ciągu

Dostęp do znaków w ciągu tekstowym

- Iteracja przez ciąg tekstowy - pętla for
- Indeksowanie elementów ciągu tekstowego

Dostęp do znaków w ciągu tekstowym

- Iteracja przez ciąg tekstowy - pętla `for`
`for zmienna in ciag_tekstowy:`
 polecenie
 polecenie
 itd.

Dostęp do znaków w ciągu tekstowym

- Iteracja przez ciąg tekstowy - pętla for

```
>>> str1 = "Programowanie"  
>>> for ch in str1:  
    print(ch)
```

Dostęp do znaków w ciągu tekstowym

- Iteracja przez ciąg tekstowy - pętla for

```
>>> str1 = "Programowanie"  
>>> for ch in str1:  
    print(ch)
```

P
r
o
g
r
a
m
o
w
a
n
i
e

>>>

Dostęp do znaków w ciągu tekstowym

```
01_zliczanie.py
File Edit Format Run Options Window Help

#Program zlicza wystąpienia litery 's'
#w podanym ciągu tekstowym.
#Program ilustruje wykorzystanie pętli for
#do uzyskania dostępu do znaków ciągu tekstowego.

def main():
    count = 0
    text = input('Podaj dowolne zdanie: ')

    for ch in text:
        if ch == 'S' or ch == 's':
            count += 1

    print('Liczba wystąpień litery S: ', count, end = '.\n')

if __name__ == '__main__':
    main()
```

Dostęp do znaków w ciągu tekstowym

- Zmienna użyta do iteracji przechowuje kopię znaków z ciągu tekstowego - jej ewentualna zmiana nie zmienia znaków w ciągu tekstowym

```
>>> str1 = "Programowanie"
>>> for ch in str1:
    ch = 'x'

>>> print(str1)
Programowanie
>>>
```

Dostęp do znaków w ciągu tekstowym

- Indeksowanie elementów ciągu tekstowego
- Wykorzystując indeks można pobrać kopię dowolnego znaku ciągu

```
>>> str1 = 'Programowanie'  
>>> ch = str1[0]  
>>> print(ch)  
P  
>>> ch = str1[3]  
>>> print(ch)  
g  
>>> ch = str1[12]  
>>> print(ch)  
e  
>>>
```

Dostęp do znaków w ciągu tekstowym

- Można wykorzystać ujemne wartości indeksów do odliczania znaków od końca

```
>>> str1 = 'Programowanie'
>>> ch = str1[-1]
>>> print(ch)
e
>>> ch = str1[-2]
>>> print(ch)
i
>>>
```

Dostęp do znaków w ciągu tekstowym

- Odniesienie do nieprawidłowego indeksu spowoduje zgłoszenie błędu **IndexError**

```
>>> str1 = 'Programowanie'
>>> str1[13]
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    str1[13]
IndexError: string index out of range
>>> str1[-14]
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    str1[-14]
IndexError: string index out of range
>>>
```

Dostęp do znaków w ciągu tekstowym

- Odniesienie do nieprawidłowego indeksu spowoduje zgłoszenie błędu **IndexError**

```
>>> faculty = "WPPT"  
>>> index = 0  
>>> while index < 5:  
    print(faculty[index])  
    index += 1
```

```
W  
P  
P  
T
```

```
Traceback (most recent call last):  
  File "<pyshell#50>", line 2, in <module>  
    print(faculty[index])  
IndexError: string index out of range  
>>>
```

Dostęp do znaków w ciągu tekstowym

- Funkcja `len()` zwraca długość ciągu znakowego

```
>>> faculty = "WPPT"  
>>> size = len(faculty)  
>>> index = 0  
>>> while index < size:  
    print(faculty[index])  
    index += 1
```

```
W  
P  
P  
T  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

```
>>> letters = 'abc'  
>>> letters += 'def'  
>>> print(letters)  
abcdef  
>>>
```

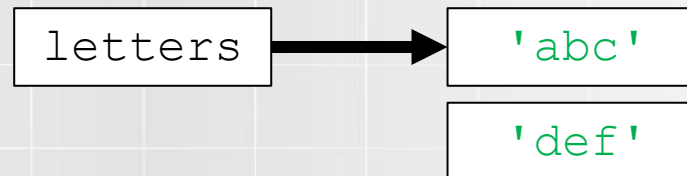


```
>>> letters = 'abc'  
>>> letters = letters + 'def'  
>>> print(letters)  
abcdef  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

```
>>> letters = 'abc'  
>>> letters += 'def'  
>>> print(letters)  
abcdef  
>>>
```

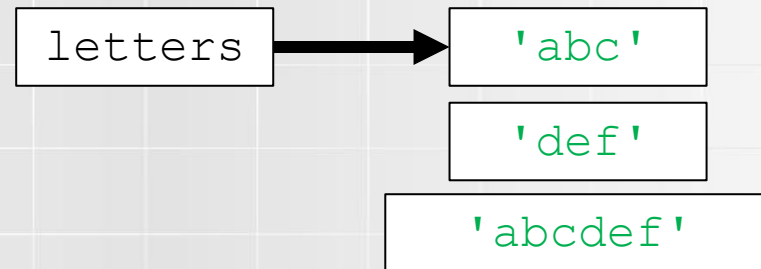


```
>>> letters = 'abc'  
>>> letters = letters + 'def'  
>>> print(letters)  
abcdef  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

```
>>> letters = 'abc'  
>>> letters += 'def'  
>>> print(letters)  
abcdef  
>>>
```

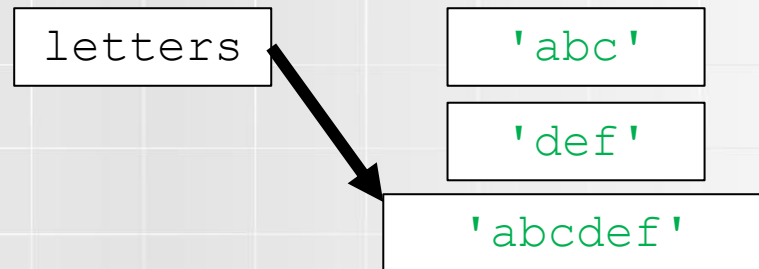


```
>>> letters = 'abc'  
>>> letters = letters + 'def'  
>>> print(letters)  
abcdef  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

```
>>> letters = 'abc'  
>>> letters += 'def'  
>>> print(letters)  
abcdef  
>>>
```



```
>>> letters = 'abc'  
>>> letters = letters + 'def'  
>>> print(letters)  
abcdef  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- Ciąg tekstowy jest niemodyfikowalny - nie można przypisać nowej wartości do elementu w istniejącym ciągu tekstowym

```
>>> name = 'Marek'
>>> name[0] = 'J'
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    name[0] = 'J'
TypeError: 'str' object does not support item assignment
>>> name = 'Kasia'
>>> name[0] = 'B'
Traceback (most recent call last):
  File "<pyshell#86>", line 1, in <module>
    name[0] = 'B'
TypeError: 'str' object does not support item assignment
>>>
```

Wycinek ciągu tekstowego

- Do kopiowania fragmentu ciągu tekstowego można wykorzystać wycinek

`ciag_tekstowy[początek:koniec]`

```
>>> str1 = 'Programowanie'
>>> str1[3:6]
'gra'
>>> str1[3:]
'gramowanie'
>>> str1[:7]
'Program'
>>> str1[-3:]
'nie'
>>> str1[:-3]
'Programowa'
>>>
```

Wycinek ciągu tekstowego

- Do kopiowania fragmentu ciągu tekstowego można wykorzystać wycinek

`ciag_tekstowy[początek:koniec:krok]`

```
>>> str1 = 'Programowanie'
>>> str1[::2]
'Pormwne'
>>> str1[::3]
'Pgmae'
>>> str1[1:-1:4]
'raa'
>>>
```

Operacje na ciągach tekstowych

- Operator `in` (`not in`) można wykorzystać do sprawdzenia, czy wskazany ciąg zawiera się (nie zawiera się) w innym

`ciag_tekstowy1 in ciag_tekstowy2`

```
>>> 'gra' in 'Programowanie'
True
>>> 'ogr' in 'Programowanie'
True
>>> 'elf' in 'Programowanie'
False
>>> 'elf' not in 'Programowanie'
True
>>>
```

Metody ciągu tekstowego

```
02_test.py
File Edit Format Run Options Window Help

#Program sprawdza jakiego rodzaju są znaki
#ciagu tekstowego podanego przez użytkownika.
#Program ilustruje działanie metod:
# isalnum(), isdigit(), isalpha(), isspace(), islower(), isupper().

def main():
    text = input('Podaj ciąg tekstowy: ')

    if text.isalnum():
        print('Ciąg tekstowy jest alfanumeryczny.')
    if text.isdigit():
        print('Ciąg tekstowy zawiera jedynie cyfry.')
    if text.isalpha():
        print('Ciąg tekstowy zawiera jedynie litery.')
    if text.isspace():
        print('Ciąg tekstowy zawiera jedynie białe znaki.')
    if text.islower():
        print('Ciąg tekstowy zawiera jedynie małe litery.')
    if text.isupper():
        print('Ciąg tekstowy zawiera jedynie duże litery.')

if __name__ == '__main__':
    main()
```

Metody ciągu tekstowego

Wybrane metody zwracające zmodyfikowaną wersję ciągu tekstowego:

- `lower()`
- `upper()`
- `lstrip()`
- `rstrip()`
- `strip()`

Metody ciągu tekstowego

Wybrane metody zwracające zmodyfikowaną wersję ciągu tekstowego:

- `lower()`
- `upper()`
- `lstrip()`
- `rstrip()`
- `strip()`

```
>>> str1 = 'Programowanie'  
>>> str1.lower()  
'programowanie'  
>>> str1.upper()  
'PROGRAMOWANIE'  
>>>
```

Metody ciągu tekstowego

Wybrane metody zwracające zmodyfikowaną wersję ciągu tekstowego:

- `lower()`
- `upper()`
- `lstrip()`
- `rstrip()`
- `strip()`

```
>>> str1 = " ,,Programowanie"
>>> str1.lstrip()
",,Programowanie"
>>> str1.rstrip()
" ,,Programowanie"
>>> str1.strip()
",,Programowanie"
>>> str1.strip(" ',")
'Programowanie'
>>>
```

Metody ciągu tekstowego

03_lower.py

File Edit Format Run Options Window Help

```
#Program ilustruje wykorzystanie metody lower()
#do sprawdzenia znaku podanego przez użytkownika.

def main():
    keep_going = 't'

    print('To jest program zadający pytanie.')
    while keep_going.lower() == 't':
        print('Czy mam powtórzyć?')
        keep_going = input("Jeśli tak, wpisz 't', w p.p. inny znak: ")

if __name__ == '__main__':
    main()
```

Metody ciągu tekstowego

Wybrane metody ciągu tekstowego do wyszukiwania i zastępowania podciągów

- `endswith()`
- `find()`
- `replace()`
- `startswith()`

Metody ciągu tekstowego

Wybrane metody ciągu tekstowego do wyszukiwania i zastępowania podciągów

- **endswith()**
- **find()**
- **replace()**
- **startswith()**

```
>>> filename = 'programowanie01.txt'
>>> filename.endswith('.txt')
True
>>>
>>> filename.endswith('.dat')
False
```

Metody ciągu tekstowego

Wybrane metody ciągu tekstowego do wyszukiwania i zastępowania podciągów

- `endswith()`
- `find()`
- `replace()`
- `startswith()`

```
>>> filename = 'programowanie01.txt'  
>>> filename.startswith('programowanie')  
True  
>>>
```

Metody ciągu tekstowego

Wybrane metody ciągu tekstowego do wyszukiwania i zastępowania podciągów

- **endswith()**
- **find()**
- **replace()**
- **startswith()**

```
>>> filename = 'programowanie01.txt'
>>> position = filename.find('01')
>>> print(position)
13
>>> position = filename.find('02')
>>> print(position)
-1
>>>
```

Metody ciągu tekstowego

Wybrane metody ciągu tekstowego do wyszukiwania i zastępowania podciągów

- `endswith()`
- `find()`
- `replace()`
- `startswith()`

```
>>> filename = 'programowanie01.txt'
>>> new_filename = filename.replace('01', '02')
>>> print(new_filename)
programowanie02.txt
>>> filename = filename.replace('.txt', '.py')
>>> print(filename)
programowanie01.py
>>>
```

Metody ciągu tekstowego

Do podzielenia ciągów tekstowych można wykorzystać metodę `split()`

```
>>> filename = 'programowanie01.txt'
>>> filename.split('.')
['programowanie01', 'txt']
>>>
```

```
>>> numbers = 'one two three four five six'
>>> numbers.split(' ')
['one', 'two', 'three', 'four', 'five', 'six']
>>> numbers.split()
['one', 'two', 'three', 'four', 'five', 'six']
>>>
```

```
>>> date_str = '26-05-2020'
>>> date_str.split('-')
['26', '05', '2020']
>>>
```

Podsumowanie (1)

- Słowniki
 - tworzenie słownika
 - operacje na słowniku - wybrane operatory, funkcje, metody
- Wykorzystanie słownika - przekazywanie argumentów
- Zbiory
 - tworzenie zbiorów
 - operacje na zbiorach - wybrane operatory, funkcje, metody

Podsumowanie (2)

- Znaki jako liczby
- Dostęp do znaków ciągu tekstowego
- Ciąg tekstowy jest niemodyfikowalny
- Wycinki ciągu tekstowego
- Wybrane metody ciągu tekstowego