

Programowanie proceduralne

INP001210WL

rok akademicki 2020/21

semestr letni

Wykład 6

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220



Plan prezentacji

- Więcej o ciągach tekstowych
 - dostęp do znaków ciągu tekstowego
 - konkatenacja ciągów tekstowych
 - ciąg tekstowy jest niemodyfikowalny
 - wycinki ciągu tekstowego
 - wybrane metody ciągu tekstowego

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

```
>>>
```

Konkatenacja ciągów tekstowych

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

```
>>> message = "Hello " + "world!"
>>> print(message)
Hello world!
>>>
```

Konkatenacja ciągów tekstowych

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

```
>>> part1 = 'Programowanie'  
>>> part2 = 'proceduralne'  
>>> total = part1 + ' ' + part2  
>>> print(total)  
Programowanie proceduralne  
>>>
```

Konkatenacja ciągów tekstowych

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

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




Konkatenacja ciągów tekstowych

- 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  
>>>
```

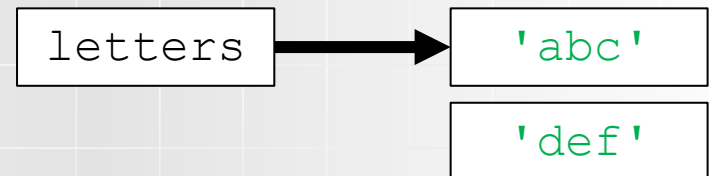


```
>>> letters = 'abc'  
>>> letters = letters + 'def'  
>>> print(letters)  
abcdef  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

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

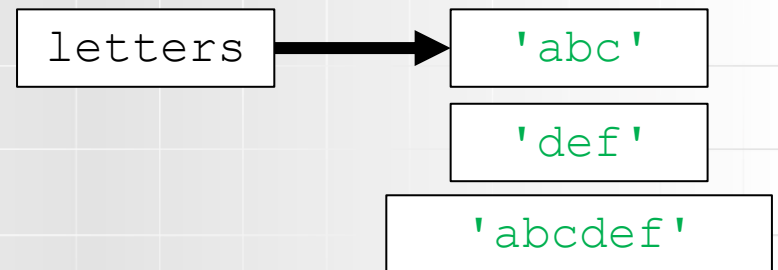


```
>>> letters = 'abc'  
>>> letters = letters + 'def'  
>>> print(letters)  
abcdef  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- Ciągi tekstowe można łączyć (konkatenować) wykorzystując operator +

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

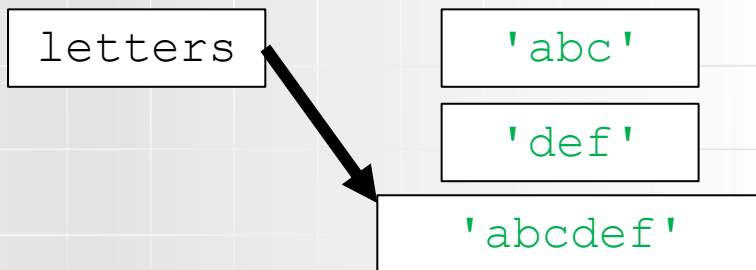


```
>>> letters = 'abc'  
>>> letters = letters + 'def'  
>>> print(letters)  
abcdef  
>>>
```

Ciąg tekstowy jest niemodyfikowalny

- 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  
>>>
```



Operator powtarzania

- Aby powtórzyć listę wielokrotnie można wykorzystać operator powtarzania *

```
>>> sound = 'Oo! '  
>>> echo = sound * 5  
>>> print(echo)  
Oo! Oo! Oo! Oo! Oo!  
>>>
```



Wycinek ciągu tekstowego

- Do kopiowania fragmentu ciągu tekstowego można wykorzystać wycinek

`ciąg_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

`ciąg_tekstowy[początek:koniec]`

```
>>> 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
#ciągu 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()`
- `rstrip()`
- `rstrip()`
- `strip()`



Metody ciągu tekstowego

Wybrane metody zwracające zmodyfikowaną wersję ciągu tekstowego:

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

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



Metody ciągu tekstowego

Wybrane metody zwracające zmodyfikowaną wersję ciągu tekstowego:

- `lower()`
- `upper()`
- `rstrip()`
- `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() >>> filename = 'programowanie01.txt'
>>> new_filename = filename.replace('01', '02')
• startswith() >>> 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

- Dostęp do znaków ciągu tekstowego
- Konkatenacja ciągów tekstowych
- Ciąg tekstowy jest niemodyfikowalny
- Wycinki ciągu tekstowego
- Wybrane metody ciągu tekstowego