Napisy, obsługa tekstów

Klasy ‘str’ i ‘bytes’

Klasy ‘str’ i ‘bytes’:

- klasa ‘str’ przechowuje jawną sekwencię wartości oznaczających symbole unikod,
- klasa ‘bytes’ przechowuje sekwencję bajtów (należy wiedzieć, co reprezentują),
- w Python’ie – wewnętrznie - wszystkie ciągi znaków traktowane są jak sekwencje unikod.

Klasy ‘str’ i ‘bytes’

Przykład 1:

```
napisStr='Zażółć gęślą jaźń'          #jawny ciąg symboli (liczb) unikod
print(napisStr)
print(type(napisStr))

napisBytes=napisStr.encode('utf8')    #ciąg bajtów - zakodowany przy pomocy utf8 ciąg symboli unikod
print(napisBytes)
print(type(napisBytes))

print(len(napisStr))
print(len(napisBytes))
```

```
#output:
Zażółć gęślą jaźń
<class 'str'>
b'Za\xc5\xbc\xc3\xb3\xc5\x82\xc4\x87 g\xc4\x99\xc5\x9bl\xc4\x85 ja\xc5\xba\xc5\x84'
<class ’bytes'>
17
26
```

Klasy ‘str’ i ‘bytes’

Przykład 2:

```
napisBytes='Zażółć gęślą jaźń'.encode('utf8')
print(napisBytes)
print(type(napisBytes))

napisStr=napisBytes.decode('utf8')
print(napisStr)
print(type(napisStr))

print(len(napisBytes))
print(len(napisStr))
```

```
#output:
b'Za\xc5\xbc\xc3\xb3\xc5\x82\xc4\x87 g\xc4\x99\xc5\x9bl\xc4\x85 ja\xc5\xba\xc5\x84'
<class 'bytes'>
Zażółć gęślą jaźń
<class 'str'>
26
17
```

Klasa ‘str’ – przydatne metody

Przykład 3a:

```
napisStr1='    TEST    '

print(napisStr1.strip())
print(napisStr1.strip().center(30))
print(napisStr1.strip().center(10))

napisStr1='TEST'
print(napisStr1.capitalize())
napisStr1='test'
print(napisStr1.capitalize())

print(napisStr1.lower())
print(napisStr1.lower().islower())

print(napisStr1.startswith('TEST'))
print(napisStr1.startswith('te'))

print(napisStr1.rjust(30))
print(napisStr1.center(30))
print(napisStr1.ljust(30))
```

```
#output

TEST
            TEST
  TEST


Test

Test

test
True

False
True

                       test
            test
test
```

Klasa ‘str’ – przydatne metody

Przykład 3b:

```
print()
napisStr2='Test test'
print(napisStr2.swapcase())
print(napisStr2.title())
print(napisStr2.title().istitle())
print('test' in napisStr2)
print(napisStr2.replace('test','----'))
```

```
#output

tEST TEST
Test Test
True
True
Test ----
```

Klasa ‘bytes’ – przydatne metody

Przykład 4a:

```
napisBytes1='    TEST    '.encode('utf8')
#klasa bytes przyjmuje ciąg bajtów
#zatem najpierw kodowanie tekstu do utf8
#lub:
napisBytes1=b'    TEST    '
#tylko dla znaków z zakresu ASCII

print(napisBytes1.strip())
print(napisBytes1.strip().center(30))
print(napisBytes1.strip().center(10))

print(napisBytes1.strip().capitalize())

print(napisBytes1.lower())
print(napisBytes1.lower().islower())
```

```
#output:
b'TEST'
b'            TEST            '
b'  TEST  '

b'    test    '

b'    test    '
True
```

Klasa ‘bytes’ – przydatne metody

Przykład 4b:

```
napisBytes1='    TEST    '.encode('utf8')

napisBytes1=napisBytes1.strip()

print(napisBytes1.startswith('TEST'.encode()))
print(napisBytes1.rjust(30))
print(napisBytes1.ljust(30))
print()
napisBytes2='Test test'.encode('utf8')

print(napisBytes2.swapcase())
print(napisBytes2.title())
print(napisBytes2.title().istitle())
print('test'.encode('utf8') in napisBytes2)
print(napisBytes2.replace('test'.encode('utf8'),'----'.encode('utf8')))
print(napisBytes2.replace(b'test',b'----'))
```

```
#output:



True
b'                    TEST'
b'TEST                    '



b'tEST TEST'
b'Test Test'
True
True
b'Test ----'
b'Test ----'
```

Notatka

Notatka:

Unicode:

- jedna liczba odpowiada jednemu znakowi (i na odwrót, nie ma powtórzeń znaków),
- definiuje obecnie 17 płaszczyzn (0-16), Plane0: 0x0 0000-0x 0 FFFF, Plane1: 0x1 0000 - 0x 1 FFFF, itd. razem daje to 1112064 tzw. punktów kodowych – znaków (niewielka część zakresu, tj. 2^11 liczb, jest zarezerwowana na potrzeby kodowania UTF-16),
- zakres/płaszczyzna podstawowa to: 0 – 65535,0x 0000-0xFFFF, zakres 0-127 pokrywa się z kodowaniem ASCII, zakres 128-255 pokrywa się z kodowaniem ISO-8859-1 (ISO/IEC 8859-1, latin-1, kodowanie znaków dla języków Europy Zachodniej),

UTF-8 (ang. 8-bit Unicode Transformation Format):

- sposób kodowania znaków unicode w od 1 do 6 bajtach (a w zasadzie od 1 do 4).

Python 2.x:

- klasa ‘str’ – ciąg bajtów (należy pamiętać co reprezentują)
- klasa ‘unicode’ –jawny ciąg (zbiór) wartości oznaczających symbole unikodu

Python 3.x - zmiana:

- klasa ‘bytes’ – ciąg bajtów (należy pamiętać co reprezentują)
- klasa ‘str’ –jawny ciąg wartości oznaczających symbole unikod

Wyrażenia regularne

Wyrażenia regularne:

- wzorce/wyrażenia opisujące łańcuchy symboli – pozwalają na wyszukiwanie mniej lub bardziej podobnych łańcuchów znaków w tekście,
- w językach programowania implementacja obsługi wyrażeń regularnych może być wbudowana (np. Perl) lub w postaci zewnętrznej biblioteki/modułu (np. Python, C++),
- obsługa wyrażeń regularnych w Python'ie odbywa się za pomocą standardowego modułu ‘re’,
- składnie różnych implementacji mogą się od siebie nieco różnić – dwie najpopularniejsze składnie to: składnia występująca w narzędziach systemów uniksowych i składnia języka Perl,
- moduł 're' oferuje składnię wyrażeń podobną do składni w Perl’u.

Moduł re

Moduł re – funkcje:

- moduł re jest modułem standardowym i zwykle jest domyślnie instalowany,
- oferuje różne funkcje/obiekty do wyszukiwania wzorców, np.: re.match(), re.search(), re.findall() oraz funkcje/obiekty pomocnicze, np.: re.compile(), re.finditer(),
- funkcje typu re.match(), re.search(), re.findall() przyjmują jako parametry wzorzec oraz badany tekst – i jeśli znajdą pasujący do wzorca fragment to zwracają tzw. 'match object'; w przeciwnym wypadku zwracają obiekt 'None',
- 'match object' przechowuje znalezione/pasujące fragmenty tekstu i są one dostępne za pomocą odpowiednich metod.

Moduł re, funkcja match()

Przykład 5 – re.match() :

```
import re

#re.match() - dopasowuje wzorzec tylko do początku tekstu
#skladnia: re.match(pattern, string, flags=0)

tekst="To jest test. To jest przykład działania wyrażeń regularnych."
matchObj=re.match('To jest', tekst)      #pasuje, zatem matchObj!=None
if matchObj:
   print(matchObj.group(0))              #zwraca dopasowany fragment
   print(matchObj[0])                    #to samo - zwraca dopasowany fragment
matchObj=re.match('jest',tekst)          #nie pasuje
if matchObj:
   print(matchObj.group(0))
matchObj=re.match('.*jest',tekst)        #pasuje
if matchObj:
   print(matchObj.group(0))
```

```
#output
To jest
To jest
To jest test. To jest
```

Moduł re, funkcja search()

Przykład 6 - re.search() :

```
import re

#re.search() - dopasowuje pierwsze wystąpienie wzorca")
#składnia: re.search(pattern, string, flags=0)

tekst="To jest test. To jest przykład działnia wyrażeń regularnych."

matchObj=re.search("jest",tekst)
if matchObj:
    print(matchObj.group(0))
#ujęcie fragmentu wzorca w nawiasy okrągłe umożliwia jego zapamiętanie w wyniku
matchObj=re.search('(jest ).*(przykład).*(wyrażeń)',tekst)
if matchObj:
    print(matchObj.groups())
    print(matchObj.group(0))
    print(matchObj.group(1))
    print(matchObj.group(2))
    print(matchObj.group(3))
```

```
#output
jest
('jest ', 'przykład', 'wyrażeń')
jest test. To jest przykład działnia wyrażeń.
jest
przykład
wyrażeń
```

Moduł re, funkcja findall()

Przykład 7 – re.findall():

```
import re

#re.findall() - dopasowuje wszystkie nie nachodzące na siebie wystąpienia wzorca
#i zwraca listę"
#składnia: re.findall(pattern, string, flags=0)

tekst="To jest test. To jest przykład działania wyrażeń regularnych."

matchObj=re.findall("jest",tekst)
if matchObj:
    print(matchObj)
matchObj=re.findall('jest.*jest.*przykład.*wyrażeń',tekst)
if matchObj:
    print(matchObj)
matchObj=re.findall('to',tekst,re.IGNORECASE)
if matchObj:
    print(matchObj)
```

```
#output:
['jest', 'jest']
['jest test. To jest przykład działania wyrażeń']
['To', 'To']
```

Moduł re - 'raw strings’

Przykład 8 – „raw strings”:

```
import re
# \n jest zamieniany przez interpreter na kod znaku nowej linii (\ to tzw. 'escape character')
tekst='To jest test raw string\n. Dalszy ciąg.'
print(tekst)
# r przed ciągiem wyłącza działanie 'escape characters' (jest to tzw. 'raw string')
tekst=r'To jest test raw string\n. Dalszy ciąg.'
print(tekst)
#to samo bez 'raw string'
tekst='To jest test raw string\\n. Dalszy ciąg.'
print(tekst)
#Python wewnętrznie wykorzystuje \ jako 'ecape character', składnia wyrażeń regularnych -
# także wykorzystuje \ jako 'escape character' – zatem mamy dublowanie się interpretacji \
# gdy korzystamy z modułu re; przykład – szukamy ciągu znaków '\n' w tekście:
matchObj=re.findall('\\\\n',tekst)        #bez 'raw string'
if matchObj:
   print(matchObj)
#z wykorzystaniem 'raw string'
matchObj=re.findall(r'\\n',tekst)
if matchObj:
   print(matchObj)
```

```
#output
To jest test raw string
. Dalszy ciąg.
To jest test raw string\n. Dalszy ciąg.
To jest test raw string\n. Dalszy ciąg.
['\\n']
['\\n']
```

Moduł re, funkcja finditer()

Przykład 9 - re.finditer():

```
import re

#re.finditer() - zwraca iterator, który daje (yields) 'match objects',
#składnia: re.finditer(pattern, string, flags=0)

tekst="To jest test. To jest przykład działania wyrażeń regularnych."
finditerObj=re.finditer(r'(\w+)',tekst)
for slowo in finditerObj:
    print(slowo.group(0))
```

```
#output:
 To
jest
test
To
jest
przykład
działania
wyrażeń
regularnych
```

Moduł re, funkcja compile()

Przykład 10 – re.compile():

```
import re
#re.compile() - zwraca obiekt wyrażeń regularnych; wykorzystuje się go, gdy wyrażenie jest długie
#składnia: regexObj=compile(pattern, flags=0)
regex_email = re.compile(              # tworzymy skomplikowane wyrażenie
   r'''(?P<adres>                      # ?P<klucz>(wzorzec) – pozwala na zapamiętywanie -
   (?P<login>[\w+.]+)                  # fragmentów w postaci słownika – {klucz:wzorzec, …}
   @
   (?P<domena>\w+(\.\w+)+)
   )''',
   re.IGNORECASE | re.VERBOSE       # flagi: ignorowanie rozmiaru liter,
)                                    # ignorowanie białych znaków – pozwala na pisanie w
                                     # wzorca w wielu liniach
tekst='mail1@gmail.com, "jan.nowak@poczta.pl"'

for match_object in regex_email.finditer(tekst):
   print(match_object.groupdict())
```

```
{'adres': 'mail1@gmail.com', 'login': 'mail1', 'domena': 'gmail.com'}
{'adres': 'jan.nowak@poczta.pl', 'login': 'jan.nowak', 'domena': 'poczta.pl'}
```

Moduł re – Match object

Przykład 11 – metoda Match.group()

```
import re

m = re.match(r"(\w+) (\w+)", "Isaac Newton, physicist")

print(m.group(0))       # Całe dopasowanie
print(m.group(1))       # Dopasowanie z pierwszego nawiasu
print(m.group(2))       # Dopasowanie z drugiego nawiasu
print(m.group(1, 2))    # Dopasowania z kilku wybranych nawiasów
print(m.groups())       # Wszystkie dopasowania - krotka
print(m.group())        # Wszystkie dopasowania - string
```

```
#output:
Isaac Newton
Isaac
Newton
('Isaac', 'Newton')
('Isaac', 'Newton')
Isaac Newton
```

Moduł re – Match object

Przykład 12 – metoda Match.group()

```
import re

m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
print(m.group('first_name'))      # Dopasowanie z pierwszego nawiasu
print(m.group('last_name'))       # Dopasowanie z drugiego nawiasu
print(m.group(1))
print(m.group(2))
```

```
#output:
Malcolm
Reynolds
Malcolm
Reynolds
```

Zadanie 1

Napisz funkcję - generator, która:

- przyjmuje dwa parametry: nazwa pliku, zmienna ‘width’,
- wyświetla tekst z podanego pliku tak, że w jednej linii nie będzie więcej niż ‘width’ znaków (przykładowy plik plik_zad1.txt).

Przykład działania (15 i 5 znaków w linii):

```
Python - język      ego.
programowania w     Jego
ysokiego poziom     skład
u ogólnego prze     nia c
znaczenia, o ro     echuj
zbudowanym paki     e się
ecie bibliotek      przze
standardowych,      jrzys
którego ideą pr     tości
zewodnią jest c     ą i z
zytelność i kla     więzł
rowność kodu źr     ością
ódłowego. Jego      . Pyt
```

Zadanie 2

Zmodyfikuj poprzedni program tak, aby:

- każdy wyraz z tekstu wyświetlany był na ekranie w osobnej linii,
- oraz wyśrodkowany w obrębie wprowadzonego rozmiaru (‘width’).

Przykład działania (dla 10 i 15 znaków w linii):

```
  Python
    -
  język
programowa
  nia
wysokiego
 poziomu
 ogólnego
przeznacze
  nia,
   o
```

```
     Python
       -
     język
 programowania
   wysokiego
     poziomu
     ogólnego
 przeznaczenia,
       o
   rozbudowanym
     pakiecie
   bibliotek
 standardowych,
     którego
      ideą
```

Zadanie 3

Napisz program, który:

- pobiera z linii komend nazwę pliku, wczytuje i wyłuskuje z niego wszystkie poprawne adresy IPv4

Przykład działania:

W plik_zad3.txt znaleziono adresy:

10.0.0.0

212.182.1.25

100.2.45.11

Zadanie 4

Napisz funkcję-walidator dla numerów PESEL, który sprawdza ich poprawność oraz wyłuskuje datę urodzenia w postaci ‘dd’-’miesiac’-'rrrr' oraz płeć. Wykorzystaj moduł **re** do 'rozbicia' numeru na części.

Przykład:
86242335618 : 23-04-2086 mezczyzna
82121199232 : 11-12-1982 mezczyzna
45122317876 : 23-12-1945 mezczyzna
98031252482 : 12-03-1998 kobieta
66022553139 : 25-02-1966 mezczyzna
66022553139z : Niedozwolony znak lub niepoprawna długość!
66022553039 : Niepoprawna suma konrtolna
660225y53039 : Niedozwolony znak lub niepoprawna długość!
166022553039 : Niedozwolony znak lub niepoprawna długość!
86323084635 : 30-12-2086 mezczyzna
63040697555 : 06-04-1963 mezczyzna
61030463386 : 04-03-1961 kobieta
74082503951 : 25-08-1974 mezczyzna
65212775511 : 27-01-2065 mezczyzna