Operacje na ciągach tekstowych w języku Python

Ciągi tekstowe są z natury niemodyfikowalne, chociaż niektóre operacje (np. konkatenacja) powodują złudzenie modyfikacji ciągu, mimo że tak naprawdę do niej nie dochodzi. Python oferuje wiele różnych narzędzi i technik programowania, które można wykorzystać do analizy i operacji na ciągach tekstowych. Czasami konieczny jest dostęp do poszczególnych znaków ciągu w celu np. weryfikacji hasła. Do analizy tekstu będziemy wykorzystywali dwie techniki, tj. pętlę `for` oraz indeksowanie.

Przykład 1.
Napisz program, który zlicza liczbę wystąpień litery T (wielkiej i małej) w tekście podanym przez użytkownika.

```
def main():
    count = 0 #zmienna licznikowa
    my_string = input('Napisz dowolne zdanie: ')

    # Zliczenie wystąpień litery T
    for ch in my_string:
        if ch == 'T' or ch == 't':
            count += 1
    print('Litera T/t wystąpiła', count, 'razy.')
main()
```

Jak wiadomo, każdy znak w ciągu tekstowym ma indeks określający jego położenie, co można wykorzystać do pobrania kopii (nie oryginału) dowolnego znaku, bo w języku Python ciąg tekstowy jest niemodyfikowalny. Przy czym indeksowanie, podobnie jak w większości języków programowania, zaczyna się od pozycji `0`, co łatwo czasami prowadzi do błędu. Jeśli zatem spróbujemy wykonać instrukcje

```
city = 'Słupsk'
index = 0
while index < 7:
    print(city[index])
    index += 1
```

to ostatnia iteracja – z powodu wykroczenia poza ciąg znaków – wywoła wyjątek `IndexError`. Bardzo łatwo uniknąć tej sytuacji, jeśli zmodyfikujemy pętlę do postaci: `while index < len(city):`. Jak wiadomo, funkcja `len()` odpowiada wyznaczenie długości ciągu tekstowego.

Często ważna jest umiejętność pobrania określonego wycinka tekstowego, zwanego również podciągiem tekstowym, którego konstrukcja jest następująca: `string[początek:koniec]`. Dwie poniższe instrukcje wyodrębniają z ciągu tekstowego imię `Julia`.

```
full_name = 'Patrycja Julia Nowak'
middle_name = full_name[9:14]
```

Gdybyśmy chcieli wyodrębnić pierwsze imię, należałoby zmodyfikować drugą instrukcję

```
first_name = full_name[:8]
```

Z kolei wyodrębnienie nazwiska wymaga instrukcji

```
last_name = full_name[15:]
```

W Pythonie używamy operatora **in** do ustalenia, czy jeden ciąg tekstowy znajduje się w innym. Przykładowo, chcemy wprowadzić jakiś tekst (z klawiatury bądź z pliku) i sprawdzić, czy znajduje się w nim jakiś ciąg znaków. Powiedzmy, że szukamy wyrazu ***student*** w zdaniu: ***Jestem studentem informatyki Akademii Pomorskiej***. Uwzględniamy przy tym wielkość liter. Możemy w tym celu napisać prosty program.

Przykład 2.

```
text1 = input("Podaj frazę do wyszukania: ")
text2 = input("Podaj tekst do przeszukania: ")
print()
if text1 in text2:
    print("Ciąg tekstowy",text1, "został znaleziony")
else:
    print("Ciąg tekstowy",text1, "nie został znaleziony")
```

Poniższy przykład pokazuje z kolei kilka metod zastosowanych do przeszukania sprawdzenia ciągu tekstowego.

Przykład 3.

```
def main():
    user_string = input('Podaj ciąg tekstowy: ')

    print('Oto kilka informacji o podanym ciągu tekstowym:')
    print()

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

Jak wiadomo, programy bardzo często muszą wyszukiwać podciągi tekstowe lub ciągi tekstowe w innych ciągach. W poniższej tabeli znajduje się kilka wybranych metod przeznaczonych do wyszukiwania podciągów tekstowych, a także metodę zastępującą wszystkie wystąpienia podciągu tekstowego.

Metoda	Opis
`startswith(podciąg_tekstowy)`	Argument `podciąg_tekstowy` jest ciągiem tekstowym. Metoda zwraca wartość `True`, jeśli ciąg tekstowy zaczyna się od argumentu `podciąg_tekstowy`.
`endswith(podciąg_tekstowy)`	Argument `podciąg_tekstowy` jest ciągiem tekstowym. Metoda zwraca wartość `True`, jeśli argument `podciąg_tekstowy` znajduje się na końcu ciągu tekstowego

`find(podciąg_tekstowy)`	Argument `podciąg_tekstowy` jest ciągiem tekstowym. Metoda zwraca najniższy indeks wskazujący położenie, w którym znaleziony został `podciąg_tekstowy`. Jeśli argument `podciąg_tekstowy` nie zostanie znaleziony, metoda zwraca wartość `-1`.
`replace(stary, nowy)`	Argument `stary` i `nowy` są ciągami tekstowymi. Metoda ta zastępuje wszystkie wystąpienia argumentu `stary` argumentem `nowy`.

Poniżej trzy bardzo proste przykłady zastosowań, których komentowanie jest zbędne.

Przykład 4.
```
filename = input('Podaj nazwę pliku: ')
if filename.endswith('.txt'):
    print('To jest nazwa pliku tekstowego.')
elif filename.endswith('.py'):
    print('To jest nazwa pliku kodu źródłowego Pythona.')
elif filename.endswith('.docx'):
    print('To jest nazwa pliku dokumentu procesora tekstu.')
else:
    print('To jest nazwa pliku nieznanego typu.')
```

Przykład 5.
```
string = 'Jestem studentem informatyki Akademii Pomorskiej'
position = string.find('student')
if position != -1:
    print('Słowo "student" zostało znalezione w położeniu o indeksie',
position)
else:
    print('Słowo "student" nie zostało znalezione.')
```

Przykład 6.
```
string = 'Jestem studentem informatyki Akademii Pomorskiej'
new_string = string.replace('informatyki', 'ekonomii')
print(new_string)
```

Python posiada także metodę `split()`, która domyślnie zwraca listę słów rozdzielonym w ciągu tekstowym spacjami.

Przykład 7.
```
def main():
    # Utworzenie ciągu tekstowego wraz z wieloma słowami
    my_string = 'jeden dwa trzy cztery'

    # Podział ciągu tekstowego
    word_list = my_string.split()

    # Wyświetlenie listy słów
    print(word_list)
main()
```

Istnieje oczywiście możliwość innego separatora, który należy podać jako argument metody `split()`.

Przykład 7.

```
def main():
    date_string = '19/05/2022'
    date_list = date_string.split('/')

    print('dzień:', date_list[0])
    print('miesiąc:', date_list[1])
    print('rok:', date_list[2])
main()
```

Nadszedł czas na krótkie podsumowanie przedstawionych wiadomości w postaci nieco obszerniejszego przykładu.

Przykład 8.

Jesteś pracownikiem działu informatyki Akademii Pomorskiej i otrzymałeś polecenie opracowania kodu programu generującego nazwy logowania (loginy) dla studentów. W loginie mają być pierwsze trzy litery imienia oraz trzy litery nazwiska studenta, a także trzy ostanie cyfry numeru albumu. Przykładowe dane studenta: `Karina Zieniewicz 044396` powinny prowadzić do automatycznego wygenerowania loginu: `KarZie396`. Ponadto należy opracować program do pobierania i weryfikacji haseł studentów. Hasło uznajemy za prawidłowe, jeśli jego długość wynosi co najmniej 8 znaków, a ciąg zawiera małe i wielkie litery alfabetu oraz jakąś cyfrę.

Poniżej znajduje się szkic aplikacji (daleko jej do doskonałości), składającej się z trzech plików: `generator_login.py`, `validate_password.py` oraz głównego programu o nazwie `login.py`. Pierwsze dwa odpowiadają za komunikację z użytkownikiem, zaś ostatni zawiera główne funkcje programu, wykorzystywane w dwóch pierwszych. W związku z tym wszystkie trzy pliki powinny znajdować się w tym samym katalogu uruchomieniowym.

Listing programu `generator_login.py`:

```
import login
def main():
    first = input('Podaj imię: ')
    last = input('Podaj nazwisko: ')
    idnumber = input('Podaj numer albumu: ')

    print('Twoja nazwa użytkownika to:')
    print(login.get_login_name(first, last, idnumber))

main()
```

Listing programu `validate_password.py`:

```
import login
def main():
    password = input('Podaj hasło: ')

    while not login.valid_password(password):
        print('Hasło jest nieprawidłowe.')
        password = input('Podaj hasło: ')

    print('Hasło jest prawidłowe.')
main()
```

Listing głównego programu o nazwie `login.py`:

```
def get_login_name(first, last, idnumber):
    # Pobranie trzech pierwszych liter imienia
    set1 = first[0:3]

    # Pobranie trzech pierwszych liter nazwiska
    set2 = last[0:3]

    # Pobranie trzech ostatnich znaków numeru albumu
    set3 = idnumber[-3:]

    # Połączenie ze sobą zbiorów znaków
    login_name = set1 + set2 + set3

    # Zwrot nazwy użytkownika
    return login_name

# Funkcja valid_password() akceptuje argument
# w postaci hasła i zwraca wartość True lub False

def valid_password(password):
    # Przypisanie wartości False zmiennym boolowskim
    correct_length = False
    has_uppercase = False
    has_lowercase = False
    has_digit = False

    # Rozpoczęcie weryfikacji hasła. Na początku
    # sprawdzana jest jego długość
    if len(password) > 7:
        correct_length = True

        # Trzeba sprawdzić każdy znak
        # i przypisać odpowiednią wartość
        # po znalezieniu wymaganego znaku
        for ch in password:
            if ch.isupper():
                has_uppercase = True
            if ch.islower():
                has_lowercase = True
            if ch.isdigit():
                has_digit = True

    # Ustalenie, czy wszystkie wymagania zostały spełnione
    if correct_length and has_uppercase and has_lowercase and has_di-
git:
        is_valid = True
    else:
        is_valid = False

    # Zwrot zmiennej is_valid
    return is_valid
```

Zadania domowe (jedno z dwóch do wyboru, a najlepiej oba)

Zadanie domowe 1.
Zmodyfikuj aplikację do generowania loginów i haseł w taki sposób, aby tworzony był plik z loginami oraz hasłami (jawnymi). W dalszym etapie należałoby weryfikować loginy i hasła studentów na podstawie informacji zawartych w tym pliku. Masz tutaj wolną rękę.

Zadanie domowe 2.
W alfabecie Morse'a litery i znaki przestankowe są zapisywane za pomocą ciągu kropek i kresek. W poniższej tabeli znajduje się fragment tego alfabetu. Opracuj program, który poprosi użytkownika o podanie ciągu tekstowego, a następnie zamieni go na ciąg tekstowy zapisany za pomocą alfabetu Morse'a.

Znak	Kod	Znak	Kod	Znak	Kod	Znak	Kod
Spacja	*przerwa*	6	−····	G	—·	Q	—·−
Przecinek	—··—	7	—···	H	····	R	·−·
Kropka	·−·−·−	8	—−··	I	··	S	···
Znak zapytania	··—··	9	——·	J	·—	T	−
0	——	A	·−	K	−·−	U	··−
1	·——	B	−···	L	·−··	V	···−
2	··—	C	−·−·	M	—	W	·—
3	···—	D	−··	N	−·	X	−··−
4	····−	E	·	O	—	Y	−·−
5	·····	F	··−·	P	·—·	Z	—··

Uwaga: w tabeli są kreski o powielanej długości, sprawiające wrażenie pojedynczych. W rzeczywistości cyfrę 9 koduje się za pomocą ciągu znaków −−−−., a literę J ciągiem .−−−.