_ Mirosław J. **Kubiak** _

PYTHON

ZADANIA Z PROGRAMOWANIA

Przykładowe funkcyjne rozwiązania

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/pyzapr
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-7255-9

Printed in Poland.

Spis treści

Rozdział 4.

Ciągi tekstowe

W tym rozdziale przedstawię zadania wraz z przykładowymi rozwiązaniami dotyczące operacji na ciągach tekstowych.

Podstawowe operacje ciągu tekstowego

Ciąg tekstowy zalicza się do typu **sekwencji**. Python oferuje wiele narzędzi i technik programowania, które można wykorzystać do analizy ciągów tekstowych i operacji na tych ciągach.

Język Python dostarcza wiele możliwości na uzyskanie dostępu do poszczególnych znaków ciągu tekstowego. Natomiast ciąg tekstowy zawiera metody umożliwiające wykonanie operacji na znakach.

W Pythonie ciąg tekstowy jest niemodyfikowalny, co oznacza że po utworzeniu ciągu nie można zmienić jego wartości.

Jednym z najłatwiejszych sposobów uzyskania dostępu do poszczególnych znaków jest użycie pętli `for` (zob. rozdział 3. w **PI**).

Zadanie

4.1 Napisz program, który iteruje ciąg tekstowy `'Krzysztof'`.

Przykładowe rozwiązanie — listing 4.1

```
# Zadanie 4.1.

def main(): # Definicja funkcji o nazwie main().

    imię = 'Krzysztof' # Ciąg tekstowy.
```

```
    for znak in imię: # Iteracja ciągu tekstowego.
        print(znak)

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.1.

Rysunek 4.1. *Efekt działania programu Zadanie 4.1*

```
K
r
z
y
s
z
t
o
f
```

Zadanie

4.2 Napisz program, który zlicza w dowolnym tekście, np. `Matematyka jest królową wszystkich nauk`, liczbę wystąpień dużej i małej litery `a`.

Przykładowe rozwiązanie — listing 4.2

```
# Zadanie 4.2.

def main(): # Definicja funkcji o nazwie main().

    ciąg_tekstowy = 'Matematyka jest królową wszystkich nauk.' # Ciąg
    ↪tekstowy.
    print("Nasz ciąg tekstowy to: ", ciąg_tekstowy, sep = "")

    print()

    licznik = 0

    for znak in ciąg_tekstowy:
        if znak == 'A' or znak == 'a':
            licznik += 1

    print("Litera a wystąpiła w tym ciągu tekstowym", licznik, "razy.")

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.2.

Rysunek 4.2. *Efekt działania programu Zadanie 4.2*

```
Nasz ciąg tekstowy to: Matematyka jest królową wszystkich
nauk.

Litera a wystąpiła w tym ciągu tekstowym 4 razy.
```

Indeksowanie to kolejny sposób uzyskania dostępu do poszczególnych znaków ciągu tekstowego. Każdy znak w takim ciągu ma indeks, który określa jego położenie. Indeksy rozpoczynają się od 0, zatem indeks pierwszego znaku to 0, drugiego 1 itd.

Zadanie

4.3 Napisz program, który dla ciągu tekstowego `Matematyka jest królową wszystkich nauk` wyświetla wszystkie indeksy oraz odpowiadające im litery.

Przykładowe rozwiązanie — listing 4.3

```
# Zadanie 4.3.

def main(): # Definicja funkcji o nazwie main().

    ciąg_tekstowy = 'Matematyka jest królową wszystkich nauk.' # Ciąg
    ↪tekstowy.

    indeks = 0

    while indeks < len(ciąg_tekstowy):
        print("Ciąg_tekstowy[", indeks, "] = ", ciąg_tekstowy[indeks],
        ↪sep = "")
        indeks += 1

main() # Wywołanie funkcji main().
```

W programie użyliśmy funkcji `len()`, która zwraca długość ciągu tekstowego.

Rezultat działania programu można zobaczyć na rysunku 4.3.

Wycinek ciągu tekstowego

Z rozdziału 2. w **PF** wiadomo, że wycinek (ang. *slice*) to pewna liczba elementów pobranych z sekwencji. Gdy pobieramy wycinek ciągu tekstowego, otrzymujemy pewną liczbę pochodzących z niego znaków, tzw. **podciąg tekstowy**. Aby pobrać wycinek ciągu tekstowego, należy utworzyć wyrażenie, którego ogólna postać jest następująca:

```
string[początek : koniec]
```

Rysunek 4.3. *Efekt działania programu Zadanie 4.3*

```
Ciąg_tekstowy[0] = M
Ciąg_tekstowy[1] = a
Ciąg_tekstowy[2] = t
Ciąg_tekstowy[3] = e
Ciąg_tekstowy[4] = m
Ciąg_tekstowy[5] = a
Ciąg_tekstowy[6] = t
Ciąg_tekstowy[7] = y
Ciąg_tekstowy[8] = k
Ciąg_tekstowy[9] = a
Ciąg_tekstowy[10] =
Ciąg_tekstowy[11] = j
Ciąg_tekstowy[12] = e
Ciąg_tekstowy[13] = s
Ciąg_tekstowy[14] = t
Ciąg_tekstowy[15] =
Ciąg_tekstowy[16] = k
Ciąg_tekstowy[17] = r
Ciąg_tekstowy[18] = ó
Ciąg_tekstowy[19] = l
Ciąg_tekstowy[20] = o
Ciąg_tekstowy[21] = w
Ciąg_tekstowy[22] = ą
Ciąg_tekstowy[23] =
Ciąg_tekstowy[24] = w
Ciąg_tekstowy[25] = s
Ciąg_tekstowy[26] = z
Ciąg_tekstowy[27] = y
Ciąg_tekstowy[28] = s
Ciąg_tekstowy[29] = t
Ciąg_tekstowy[30] = k
Ciąg_tekstowy[31] = i
Ciąg_tekstowy[32] = c
Ciąg_tekstowy[33] = h
Ciąg_tekstowy[34] =
Ciąg_tekstowy[35] = n
Ciąg_tekstowy[36] = a
Ciąg_tekstowy[37] = u
Ciąg_tekstowy[38] = k
Ciąg_tekstowy[39] = .
```

Indeks pierwszego znaku to `początek`, natomiast `koniec` to indeks oznaczający jego zakończenie. Wyrażenie to zwróci kopię znaków od indeksu `początek` aż do indeksu `koniec`, przy czym znak o indeksie `koniec` **nie znajduje się** w tym wycinku.

Na przykład:

```
litery = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
wycinek = litery[0 : 7]
```

Spróbuj odgadnąć, jakie litery zawiera `wycinek`. Rozwiązanie znajdziesz w zadaniu 4.4.

Zadanie

4.4 Napisz program, który z zadanego ciągu tekstowego `'ABCDEFGHIJKLMNOPQ↪RSTUVWXYZ'` pobiera pewien dowolny wycinek.

Przykładowe rozwiązanie — listing 4.4

```
# Zadanie 4.4.

def main(): # Definicja funkcji o nazwie main().

    litery = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' # Ciąg tekstowy.

    print("Nasz ciąg tekstowy to: ", litery, ".", sep = "")

    wycinek = litery[0 : 7] # Wycinek.

    print("Nasz wycinek to: ", wycinek, ".", sep = "")

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.4.

Rysunek 4.4. *Efekt działania programu Zadanie 4.4*

```
Nasz ciąg tekstowy to: ABCDEFGHIJKLMNOPQRSTUVWXYZ.
Nasz wycinek to: ABCDEFG.
```

Język Python oferuje operatory i metody przeznaczone do wyszukiwania i pobierania zmodyfikowanych kopii ciągu tekstowego. Omówię je za pomocą prostych zadań z przykładowymi rozwiązaniami.

Zadanie

4.5 Napisz program, który testuje ciąg tekstowy za pomocą operatorów `in` i `not in`.

Przykładowe rozwiązanie — listing 4.5

```
# Zadanie 4.5.

def main(): # Definicja funkcji o nazwie main().
```

```
    imiona = 'Małgorzata Jakub Bartosz Julia Max Agata' # Ciąg tekstowy.

    print("Ciąg tekstowy: ", imiona, ".", sep = "")

    print()

    if 'Jakub' in imiona:
        print("Jakub został znaleziony.")

    print()

    if 'Adam' not in imiona:
        print("Adam nie został znaleziony.")

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.5.

Rysunek 4.5. *Efekt działania programu Zadanie 4.5*

```
Ciąg tekstowy: Małgorzata Jakub Bartosz Julia Max Agata.

Jakub został znaleziony.

Adam nie został znaleziony.
```

Metody sprawdzające ciąg tekstowy

Python oferuje kilka metod, które umożliwiają sprawdzenie zawartości ciągu tekstowego. Tabela 4.1 zawiera opis tych metod.

Zadanie

4.6 Napisz program, który ilustruje przykładowe metody z tabeli 4.1, sprawdzające zawartość ciągu tekstowego.

Przykładowe rozwiązanie — listing 4.6

```
# Zadanie 4.6.

def main(): # Definicja funkcji o nazwie main().

   ciąg_tekstowy = 'CDE' # Przykładowy ciąg tekstowy.

   if ciąg_tekstowy.isalnum():
       print("Ciąg tekstowy", ciąg_tekstowy, "jest alfanumeryczny.")

   if ciąg_tekstowy.isdigit():
       print("Ciąg tekstowy", ciąg_tekstowy, "zawiera tylko cyfry.")
```

Tabela 4.1. *Wybrane metody sprawdzające zawartość ciągu tekstowego [1]*

Metoda	Opis
`isalnum()`	Metoda ta zwraca wartość `True`, gdy ciąg tekstowy zawiera **tylko** litery lub cyfry i ma wielkość co najmniej jednego znaku. W przypadku przeciwnym zwraca wartość `False`.
`isalpha()`	Metoda ta zwraca wartość `True`, gdy ciąg tekstowy zawiera **tylko** litery i ma wielkość co najmniej jednego znaku. W przypadku przeciwnym zwraca wartość `False`.
`isdigit()`	Metoda ta zwraca wartość `True`, gdy ciąg tekstowy zawiera **tylko** cyfry i ma wielkość co najmniej jednego znaku. W przypadku przeciwnym zwraca wartość `False`.
`islower()`	Metoda ta zwraca wartość `True`, gdy ciąg tekstowy zawiera same małe litery i składa się z przynajmniej jednej litery. W przypadku przeciwnym zwraca wartość `False`.
`isspace()`	Metoda ta zwraca wartość `True`, gdy ciąg tekstowy zawiera same białe znaki[1] i ma wielkość co najmniej jednego znaku. W przypadku przeciwnym zwraca wartość `False`.
`isupper()`	Metoda ta zwraca wartość `True`, gdy ciąg tekstowy zawiera same duże litery i składa się z przynajmniej z jednej litery. W przypadku przeciwnym zwraca wartość `False`.

```
    if ciąg_tekstowy.isalpha():
        print("Ciąg tekstowy", ciąg_tekstowy, "zawiera tylko znaki
        ↪alfabetu.")

    if ciąg_tekstowy.isspace():
        print("Ciąg tekstowy", ciąg_tekstowy, "zawiera tylko białe
        ↪znaki.")

    if ciąg_tekstowy.islower():
        print("Ciąg tekstowy", ciąg_tekstowy, "zawiera tylko małe znaki.")

    if ciąg_tekstowy.isupper():
        print("Ciąg tekstowy", ciąg_tekstowy, "zawiera tylko duże znaki.")

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.6.

[1] Białe znaki to m.in.: spacja, znak nowego wiersza (`\n`) i tabulator (`\t`).

Rysunek 4.6. *Efekt działania programu Zadanie 4.6*

```
Ciąg tekstowy CDE jest alfanumeryczny.
Ciąg tekstowy CDE zawiera tylko znaki alfabetu.
Ciąg tekstowy CDE zawiera tylko duże znaki.
```

Zachęcam Cię do przetestowania powyższego programu dla dowolnego ciągu tekstowego.

Metody modyfikujące ciąg tekstowy

Wcześniej wspomniałem, że ciąg tekstowy jest niemodyfikowalny. Istnieją metody, które zwracają zmodyfikowaną wersję ciągu tekstowego, np. zamiana dużych liter na małe litery. Niektóre z nich zostały przedstawione w zadaniach poniżej.

Zadanie 4.7 Napisz program, który w ciągu tekstowym zamienia małe litery na duże litery.

Przykładowe rozwiązanie — listing 4.7

```
# Zadanie 4.7.

def main(): # Definicja funkcji o nazwie main().

   ciąg_tekstowy = 'katarzyna nowak' # Przykładowy ciąg tekstowy.

   print("Małe litery ", ciąg_tekstowy, " zostały zamienione na duże
   ↪litery ", ciąg_tekstowy.upper(), ".", sep = "")

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.7.

Rysunek 4.7. *Efekt działania programu Zadanie 4.7*

```
Małe litery katarzyna nowak zostały zamienione na duże
litery KATARZYNA NOWAK.
```

W programie do zamiany małych liter na duże litery skorzystaliśmy z metody `upper()`. W ramach samodzielnych ćwiczeń dokonaj odwrotnej zamiany, tj. dużych liter na małe litery, używając metody `lower()`. Metody `upper()` i `lower()` są bardzo przydatne podczas porównywania ciągów tekstowych bez względu na wielkość liter.

Inne metody modyfikujące ciąg tekstowy zostały przedstawione w oficjalnej dokumentacji dotyczącej języka Python na stronie *https://www.python.org/*. Zachęcam, byś w ramach ćwiczeń zapoznał się z tymi metodami.

Zadanie

4.8 Napisz program, który tworzy w systemie bezpieczne hasło [1]. Powinno ono spełniać następujące wymagania:

- musi składać się przynajmniej z siedmiu znaków;
- musi zawierać przynajmniej jedną dużą literę;
- musi zawierać przynajmniej jedną małą literę;
- musi zawierać przynajmniej jedną cyfrę.

Po zdefiniowaniu hasła powinno ono zostać sprawdzone — czy spełnia nasze wymagania.

Przykładowe rozwiązanie składające się z dwóch plików: Moduł login.py i Program główny.py — listing 4.8

```
# Moduł login.py.

def get_login_name(pierwszy, ostatni, nr_ident):
    set1 = pierwszy[0 : 3] # Pobranie trzech pierwszych liter imienia.

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

    set3 = nr_ident[-3 : ] # Pobranie trzech ostatnich znaków numeru
    ↪identyfikacyjnego.

    login_name = set1 + set2 + set3 # Konkatenacja.

    return login_name

def ważne_hasło(hasło): # Ustalenie wartości początkowych.
    correct_length = False
    has_uppercase = False
    has_lowercase = False
    has_digits = False

    if len(hasło) >= 7: # Sprawdzenie długości hasła.
        correct_length = True

    for ch in hasło:
        if ch.isupper(): # Sprawdzenie, czy jest duża litera.
            has_uppercase = True
        if ch.islower(): # Sprawdzenie, czy jest mała litera.
```

```
            has_lowercase = True
        if ch.isdigit(): # Sprawdzenie, czy jest cyfra.
            has_digit = True

    # Ustalenie, czy wszystkie wymagania zostały spełnione.
    if correct_length and has_uppercase and has_lowercase and has_digit:
        jest_ważne = True
    else:
        jest_ważne = False

    return jest_ważne
```

Program główny

```
# Zadanie 4.8.

import login # Importujemy moduł login.

def main(): # Definicja funkcji o nazwie main().

    hasło = input("Podaj hasło: ")

    if not login.ważne_hasło(hasło):
        print("Hasło jest nieprawidłowe.")
    else:
        print("OK. Hasło jest prawidłowe.")

main() # Wywołanie funkcji main().
```

Program główny zostanie wykonany tylko wtedy, jeśli dołączymy do niego moduł `login.py`, który powinien się znaleźć w tym samym katalogu co program główny.

Rezultat działania programu można zobaczyć na rysunku 4.8.

Rysunek 4.8. *Efekt działania programu Zadanie 4.8*

```
Podaj hasło: Krzysztof123
OK. Hasło jest prawidłowe.
```

Operator powtarzania

Zadanie

4.9 Napisz program, który wyświetla dziewięć wierszy składających się z litery `"0"`, z których każdy jest coraz dłuższy, i dziewięć takich samych wierszy, z których każdy jest coraz krótszy.

Przykładowe rozwiązanie — listing 4.9

```
# Zadanie 4.9.

def main(): # Definicja funkcji o nazwie main().

    for i in range(1, 10): # Wyświetlenie coraz dłuższych wierszy.
        print("0" * i)

    for i in range(8, 0, -1): # Wyświetlenie coraz krótszych wierszy.
        print("0" * i)

main() # Wywołanie funkcji main().
```

W programie wykorzystano **operator powtarzania**, którego ogólna postać jest następująca:

```
ciąg_tekstowy_do_skopiowania * n
```

gdzie `n` jest liczbą powtórzeń.

Rezultat działania programu można zobaczyć na rysunku 4.9.

Rysunek 4.9. *Efekt działania programu Zadanie 4.9*

```
0
00
000
0000
00000
000000
0000000
00000000
000000000
00000000
0000000
000000
00000
0000
000
00
0
```

Metoda split()

Metoda `split()` używa spacji jako separatorów. Zwraca listę słów, które w ciągu tekstowym są rozdzielone spacjami.

Zadanie

4.10 Napisz program, który korzystając z metody `split()`, dokonuje podziału ciągu tekstowego zawierającego imiona: `'Adam Ewa Janusz Dorota'`.

Przykładowe rozwiązanie — listing 4.10

```
# Zadanie 4.10.

def main(): # Definicja funkcji o nazwie main().
    ciąg_tekstowy = 'Adam Ewa Janusz Dorota' # Przykładowy ciąg tekstowy.
    lista_słów = ciąg_tekstowy.split()
    print(lista_słów)

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.10.

Rysunek 4.10. *Efekt działania programu Zadanie 4.10*

```
['Adam', 'Ewa', 'Janusz', 'Dorota']
```

Zadanie

4.11 Napisz program, który korzystając z metody `split('/')`, dokonuje podziału ciągu tekstowego zawierającego przykładową datę: `'10/06/2020'`.

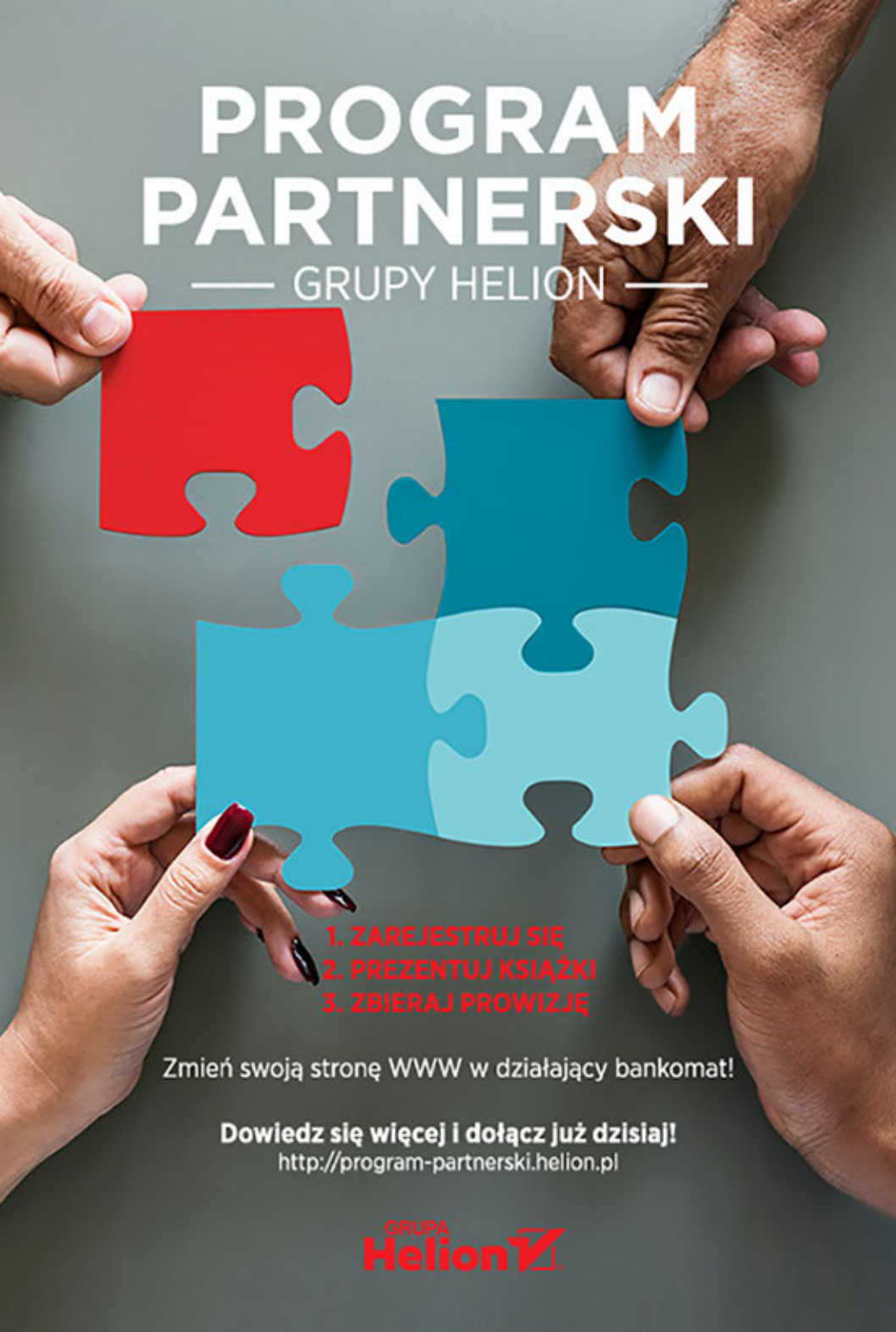
Przykładowe rozwiązanie — listing 4.11

```
# Zadanie 4.11.

def main(): # Definicja funkcji o nazwie main().
    ciąg_tekstowy = '10/06/2020' # Przykładowy ciąg tekstowy.
    lista_daty = ciąg_tekstowy.split('/')
    print("Dzień: ", lista_daty[0], ".", sep = "")
    print("Miesiąc: ", lista_daty[1], ".", sep = "")
    print("Rok: ", lista_daty[2], ".", sep = "")

main() # Wywołanie funkcji main().
```

Rezultat działania programu można zobaczyć na rysunku 4.11.

Rysunek 4.11. *Efekt działania programu Zadanie 4.11*

```
Dzień: 10.
Miesiąc: 06.
Rok: 2020.
```


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

PROGRAMUJ FUNKCYJNIE W PYTHONIE!

- **Poznaj funkcyjny paradygmat programowania**
- **Naucz się wykorzystywać go w języku Python**
- **Rozwiązuj praktyczne problemy programistyczne**

Python to obecnie jeden z najpopularniejszych języków programowania, a jego znajomość zapewnia zatrudnienie w największych firmach i przy najciekawszych projektach w branży informatycznej. Szerokie możliwości, duża elastyczność i wszechstronność, przejrzystość i zwięzłość składni, czytelność i klarowność kodu, rozbudowany pakiet bibliotek standardowych, niemal nieograniczone zastosowanie w różnych dziedzinach nauki i biznesu — wszystko to sprawia, że język ten z pewnością utrzyma swoją pozycję, a programujące w nim osoby jeszcze długo będą należały do najbardziej pożądanych specjalistów na rynku IT.

Jedną z niewątpliwych zalet Pythona jest to, że wspiera różne paradygmaty programowania, w tym wydajne programowanie funkcyjne. Jeśli chcesz poszerzyć swoją wiedzę na temat języka i dowiedzieć się, jak wykorzystać jego możliwości w tym podejściu, sięgnij po książkę *Python. Zadania z programowania. Przykładowe funkcyjne rozwiązania*. Dzięki zamieszczonym w niej zadaniom o różnym poziomie trudności oraz ich rozwiązaniom poznasz podstawy programowania funkcyjnego w Pythonie i nauczysz się pisać zwięzłe i eleganckie programy, które działają szybko i zużywają mniej zasobów, praktyczne wskazówki zaś pomogą Ci zrozumieć bardziej zawiłe zagadnienia.

- **Wprowadzenie do programowania funkcyjnego w Pythonie**
- **Sekwencyjne struktury danych i operacje na plikach**
- **Wykorzystanie krotek i ciągów tekstowych**
- **Zastosowanie słowników i zbiorów**
- **Wybrane moduły programowania funkcyjnego**
- **Funkcje rekurencyjne i rekurencja ogonowa**
- **Programowanie synchroniczne i asynchroniczne**
- **Podstawy programowania współbieżnego i równoległego**

Przekonaj się, jak prosty może być język Python!

Jeśli chcesz poznać podstawy języka Python oraz opanować paradygmaty imperatywny i obiektowy, przed lekturą tej książki sięgnij po inną publikację tego autora:

Python. Zadania z programowania. Przykładowe imperatywne rozwiązania

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-7255-9

Cena: 39,90 zł