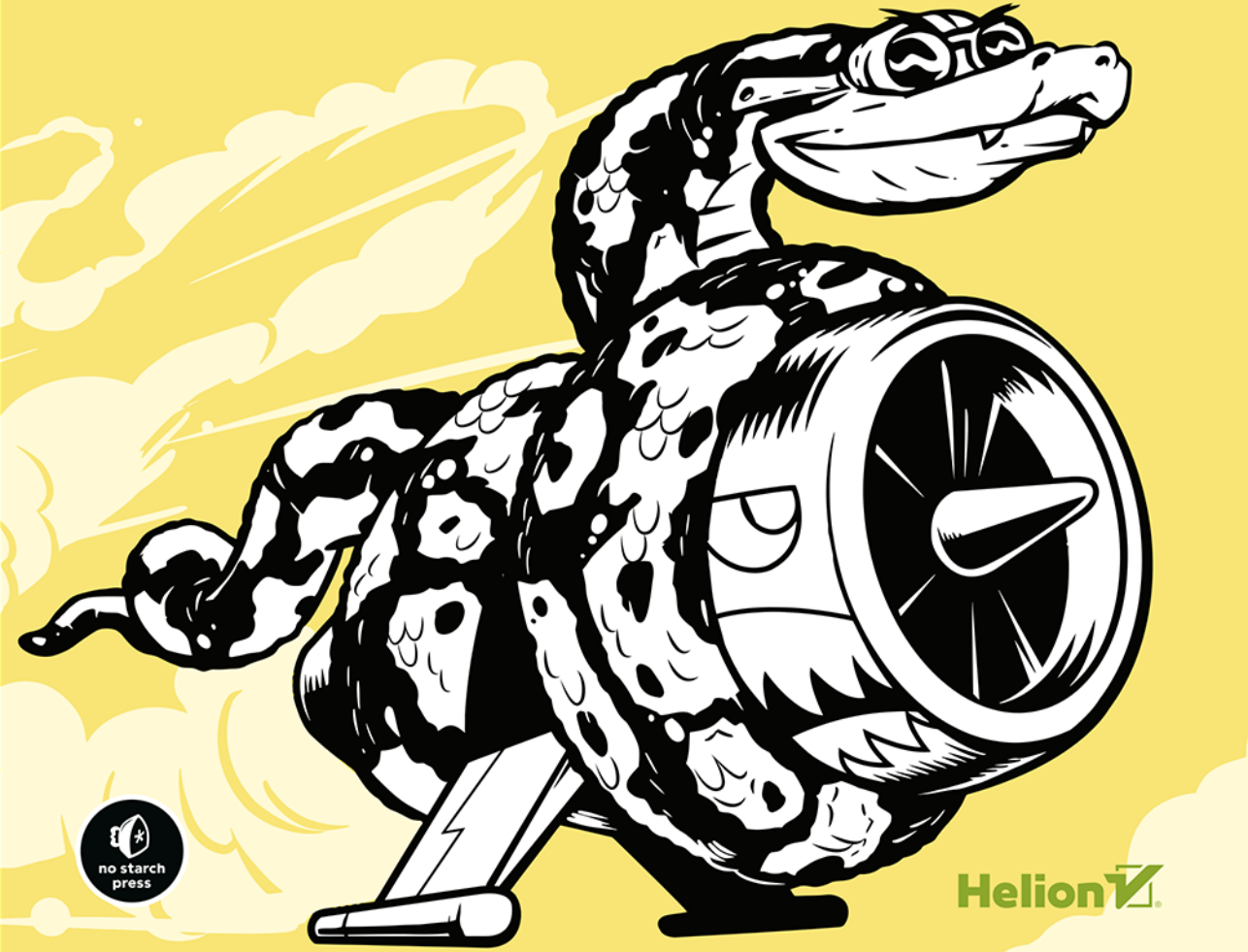
WYDANIE II

PYTHON

INSTRUKCJE DLA PROGRAMISTY

ERIC MATTHES



no starch press

Helion

Tytuł oryginału: Python Crash Course: A Hands-On,
Project-Based Introduction to Programming, 2nd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-6360-1

Helion SA
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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/blkpy2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/blkpy2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

CZĘŚĆ II. PROJEKTY

6

Słowniki

W TYM ROZDZIALE DOWIESZ SIĘ, JAK UŻYWAĆ W PYTHONIE SŁOWNIKÓW, KTÓRE POZWALAJĄ POŁĄCZYĆ POWIĄZANE ZE SOBĄ INFORMACJE. ZOBACZYSZ, JAK UZYSKAĆ DOSTĘP DO danych znajdujących się w słowniku oraz jak modyfikować te dane. Ponieważ słowniki mogą przechowywać praktycznie nieograniczoną ilość informacji, zaprezentuję iterację przez dane umieszczone w słowniku. Ponadto nauczysz się zagnieżdżać słowniki wewnątrz list, listy wewnątrz słowników, a nawet słowniki wewnątrz innych słowników.

Poznanie słowników pozwoli Ci znacznie wierniej modelować różne rzeczywiste obiekty. Zyskasz możliwość utworzenia słownika przedstawiającego osobę i przechowującego wszystkie informacje o tej osobie. Będziesz mógł na przykład przechowywać takie dane jak imię i nazwisko, wiek, miejsce zamieszkania, zawód, a także wszelkie inne dane opisujące tę osobę. Ponadto będziesz mógł przechowywać dwa dowolne rodzaje informacji, które będą do siebie dopasowane, na przykład listę słów i ich znaczenie, listę osób i ich ulubione liczby, listę szczytów i ich wysokości.

Prosty słownik

Rozważ grę, w której występują obcy o różnych kolorach, a liczba punktów uzyskiwanych po zestrzeleniu obcego jest zależna od jego koloru. Poniżej przedstawiłem słownik przeznaczony do przechowywania informacji o obcym.

Plik alien.py:

```
alien_0 = {'color': 'zielony', 'points': 5}

print(alien_0['color'])
print(alien_0['points'])
```

W słowniku `alien_0` przechowujemy kolor obcego oraz liczbę punktów otrzymywanych za jego unicestwienie. Dwa wywołania `print()` uzyskują dostęp do słownika i wyświetlają przechowywane w nim informacje:

```
zielony
5
```

Podobnie jak to jest w przypadku większości nowych koncepcji w programowaniu, przywyknięcie do słowników wymaga praktyki. Kiedy nabędziesz nieco doświadczenia w pracy ze słownikami, przekonasz się, jak można efektywnie wykorzystywać je do modelowania rzeczywistych sytuacji.

Praca ze słownikami

W Pythonie *słownik* jest kolekcją *par klucz-wartość*. Każdy *klucz* jest połączony z wartością, za pomocą klucza można uzyskać dostęp do powiązanej z nim wartości. Wartością klucza może być liczba, ciąg tekstowy, lista, lub nawet inny słownik. W rzeczywistości wartością słownika może być dowolny obiekt możliwy do utworzenia w Pythonie.

Słownik w Pythonie jest opakowany w nawias klamrowy i zawiera serię par klucz-wartość, tak jak pokazałem w poprzednim przykładzie:

```
alien_0 = {'color': 'zielony', 'points': 5}
```

Para klucz-wartość to zbiór wartości powiązanych ze sobą. Kiedy podajesz klucz, Python zwraca powiązaną z nim wartość. Połączenie klucza z wartością odbywa się za pomocą dwukropka, a poszczególne pary klucz-wartość są rozdzielone przecinkami. W słowniku można przechowywać dowolną liczbę par klucz-wartość.

Najprostszy słownik ma dokładnie jedną parę klucz-wartość, tak jak pokazałem poniżej w zmodyfikowanej wersji słownika `alien_0`:

```
alien_0 = {'color': 'zielony'}
```

Ten słownik przechowuje jeden fragment informacji dotyczący obcego, a dokładnie jego kolor. W omawianym słowniku ciąg tekstowy `'color'` jest kluczem, z którym jest powiązana wartość `'zielony'`.

Uzyskiwanie dostępu do wartości słownika

Aby pobrać wartość powiązaną z kluczem, należy podać nazwę słownika oraz nazwę klucza ujętą w nawias kwadratowy:

```
alien_0 = {'color': 'zielony'}
print(alien_0['color'])
```

Wywołanie `print()` wyświetla wartość klucza `'color'` w słowniku `alien_0`:

```
zielony
```

Słownik może zawierać nieograniczoną liczbę par klucz-wartość. Przykładowo poniżej znajduje się początkowy słownik `alien_0` z dwiema parami klucz-wartość:

```
alien_0 = {'color': 'zielony', 'points': 5}
```

Teraz można uzyskać dostęp do koloru obcego oraz liczby punktów przyznawanych za jego zestrzelenie. Jeżeli gracz zestrzeli obcego, liczbę punktów, które należy mu przyznać, można sprawdzić za pomocą kodu podobnego do poniższego:

```
alien_0 = {'color': 'zielony', 'points': 5}

new_points = alien_0['points'] ❶
print(f"Zdobyłeś {new_points} punktów!") ❷
```

Odwołując się do zdefiniowanego słownika, kod w wierszu ❶ pobiera wartość przypisaną do klucza `'points'`. Następnie ta wartość zostaje umieszczona w zmiennej `new_points`. Kod w wierszu ❷ konwertuje liczbę całkowitą do postaci ciągu tekstowego i wyświetla komunikat o liczbie punktów zdobytych przez gracza:

```
Zdobyłeś 5 punktów!
```

Jeżeli powyższy kod będzie wykonywany po każdym zestrzeleniu obcego, będziesz mógł pobierać liczbę punktów przyznawanych za unicestwienie danego obcego.

Dodanie nowej pary klucz-wartość

Słownik to struktura dynamiczna, więc nowe pary klucz-wartość można dodawać w każdej chwili. W celu dodania nowej pary klucz-wartość należy podać nazwę słownika, ujętą w nawias kwadratowy nazwę nowego klucza oraz wartość przypisywaną do danego klucza.

Do przedstawionego wcześniej słownika `alien_0` dodamy teraz dwie nowe informacje: współrzędne *X* i *Y* położenia obcego, co ułatwi nam jego wyświetlenie w odpowiednim miejscu na ekranie. Umieścimy teraz obcego przy lewej krawędzi, w odległości 25 pikseli od górnej krawędzi ekranu. Ponieważ współrzędne ekranu zwykle rozpoczynają się w lewym górnym rogu, w celu umieszczenia obcego przy lewej krawędzi należy współrzędnej X przypisać wartość 0, natomiast jego odsunięcie o 25 pikseli od górnej krawędzi ekranu wymaga przypisania współrzędnej Y wartości 25, tak jak przedstawiłem w poniższym fragmencie kodu:

Plik alien.py:

```
alien_0 = {'color': 'zielony', 'points': 5}
print(alien_0)

alien_0['x_position'] = 0 ❶
alien_0['y_position'] = 25 ❷
print(alien_0)
```

Rozpoczynamy od zdefiniowania takiego samego słownika jak wcześniej. Następnie wyświetlamy jego zawartość, aby w ten sposób mieć punkt odniesienia. W wierszu ❶ dodajemy do słownika nową parę klucz-wartość: klucz `'x_position'`, wartość 0. Z kolei w wierszu ❷ dodajemy do słownika drugą nową parę: tym razem klucz to `'y_position'`, natomiast wartość to 25. Po ponownym wyświetleniu zmodyfikowanego słownika można dostrzec, że nowe pary klucz-wartość faktycznie zostały w nim umieszczone:

```
{'color': 'zielony', 'points': 5}
{'color': 'zielony', 'points': 5, 'y_position': 25, 'x_position': 0}
```

Ostateczna wersja słownika zawiera cztery pary klucz-wartość. Dwie początkowe określają kolor obcego i liczbę punktów przyznawanych za jego zestrzelenie. Z kolei dwie nowe pary przechowują informacje o położeniu obcego na ekranie.

UWAGA *Począwszy od wydania Python 3.7, słownik zachowuje kolejność, w której były dodawane pary klucz-wartość. Podczas wyświetlania słownika lub iteracji przez jego elementy zostaną one wyświetlone w kolejności dodawania ich do słownika.*

Rozpoczęcie pracy od pustego słownika

Czasami będzie wygodne lub wręcz konieczne rozpoczęcie pracy od pustego słownika, do którego dopiero później będą wstawiane pary klucz-wartość. Aby rozpocząć wypełnianie pustego słownika, zdefiniuj go za pomocą pustego nawiasu klamrowego, a następnie dodawaj poszczególne pary klucz-wartość, po jednej w każdym wierszu. Poniżej pokazałem budowanie słownika `alien_0` z zastosowaniem tego rodzaju podejścia:

Plik alien.py:

```
alien_0 = {}

alien_0['color'] = 'zielony'
alien_0['points'] = 5

print(alien_0)
```

W powyższym fragmencie kodu zdefiniowaliśmy pusty słownik `alien_0`, do którego później dodaliśmy informacje o kolorze obcego i punktach przyznawanych za jego zestrzelenie. Wynikiem jest powstanie słownika dokładnie takiego samego jak we wcześniejszych przykładach:

```
{'color': 'zielony', 'points': 5}
```

Pusty słownik tworzymy najczęściej wtedy, kiedy chcemy przechowywać dane dostarczane przez użytkownika lub mamy do czynienia z kodem, który automatycznie generuje ogromną liczbę par klucz-wartość.

Modyfikowanie wartości słownika

Aby zmodyfikować wartość w słowniku, należy podać jego nazwę, ujętą w nawias kwadratowy nazwę klucza oraz nową wartość, która ma zostać przypisana do wskazanego klucza. Rozważmy na przykład sytuację, gdy w trakcie gry obcy zmienia kolor z zielonego na żółty:

Plik alien.py:

```
alien_0 = {'color': 'zielony'}
print(f"Obcy ma kolor {alien_0['color']}.")

alien_0['color'] = 'żółty'
print(f"Obcy ma teraz kolor {alien_0['color']}.")
```

Zaczynamy od zdefiniowania słownika `alien_0` zawierającego jedynie informację o kolorze obcego. Następnie wartość klucza `'color'` zmieniamy z `'zielony'` na `'żółty'`. Wygenerowane dane wyjściowe pokazują, że kolor obcego faktycznie został zmieniony z zielonego na żółty:

```
Obcy ma kolor zielony.
Obcy ma teraz kolor żółty.
```

Bardziej interesującym przykładem może być monitorowanie położenia obcego, który porusza się z różną szybkością. W słowniku przechowujemy wartość określającą bieżącą szybkość obcego i używamy jej do ustalenia odległości, jaką powinien pokonać obcy poruszający się w prawą stronę:

```
alien_0 = {'x_position': 0, 'y_position': 25, 'speed': 'średnio'}
print(f"Początkowa wartość x-position: {alien_0['x_position']}")

# Przesunięcie obcego w prawo.
# Ustalenie odległości, jaką powinien pokonać obcy poruszający się z daną szybkością.
if alien_0['speed'] == 'wolno': ❶
    x_increment = 1
elif alien_0['speed'] == 'średnio':
    x_increment = 2
else:
    # To musi być szybki obcy.
    x_increment = 3

# Nowe położenie to suma dotychczasowego położenia i wartości x_increment.
alien_0['x_position'] = alien_0['x_position'] + x_increment ❷

print(f"Nowa wartość x-position: {alien_0['x_position']}")
```

Rozpoczynamy od zdefiniowania obcego wraz z początkowym położeniem *X* i *Y*, a także szybkością określoną jako `'średnio'`. W celu zachowania prostoty przykładu pomijamy wartości koloru i przyznawanych punktów, ale ten przykład oczywiście będzie wyglądał dokładnie tak samo po uwzględnieniu wspomnianych danych. Wyświetlamy pierwotną wartość `x_position`, aby pokazać, o jaką odległość w prawą stronę przesunął się obcy.

W wierszu ❶ konstrukcja `if-elif-else` pozwala ustalić odległość, jaką powinien pokonać obcy przesuwający się w prawą stronę, i przechowuje ją w zmiennej `x_increment`. Jeżeli szybkość obcego jest określona jako `'wolno'`, przesunie się on tylko o jedną jednostkę w prawo. Szybkość `'średnio'` powoduje przesunięcie się o dwie jednostki w prawo, natomiast `'szybko'` o trzy. Kiedy zostanie ustalona odległość do przebycia, w wierszu ❷ do aktualnej wartości klucza `x_position` dodajemy wartość zmiennej `x_increment`, a sumę umieszczamy w kluczu `x_position` słownika.

Ponieważ mamy do czynienia z obcym poruszającym się ze średnią szybkością, przesunie się on o dwie jednostki w prawo:

```
Początkowa wartość x-position: 0
Nowa wartość x-position: 2
```

Ta technika jest całkiem dobra: dzięki zmianie jednej wartości w słowniku opisującym obcego możemy zmienić też jego ogólne zachowanie. Na przykład poniższe polecenie sprawia, że nasz średnio szybki obcy zaczyna poruszać się szybko:

```
alien_0['speed'] = szybko
```

W trakcie następnego wykonywania kodu blok konstrukcji `if-elif-else` przypisze zmiennej `x_increment` większą wartość.

Usuwanie pary klucz-wartość

Kiedy przechowywany w słowniku fragment informacji nie jest dłużej potrzebny, za pomocą polecenia `del` można całkowicie usunąć parę klucz-wartość. Do prawidłowego działania polecenie `del` potrzebuje mieć podaną nazwę słownika oraz klucz przeznaczony do usunięcia.

Na przykład ze słownika `alien_0` chcemy usunąć klucz `'points'` wraz z jego wartością:

Plik alien.py:

```
alien_0 = {'color': 'zielony', 'points': 5}
print(alien_0)

del alien_0['points'] ❶
print(alien_0)
```

Polecenie w wierszu ❶ nakazuje Pythonowi usunięcie klucza `'points'` ze słownika `alien_0`, a także wartości powiązanej z wymienionym kluczem. Wygenerowane dane wyjściowe pokazują, że klucz `'points'` i jego wartość `5` zostały usunięte, natomiast pozostała część słownika nie została zmieniona:

```
{'color': 'zielony', 'points': 5}
{'color': 'zielony'}
```

UWAGA *Należy pamiętać, że operacja usunięcia pary klucz-wartość jest nieodwracalna.*

Słownik podobnych obiektów

W poprzednim przykładzie przechowywaliśmy różne rodzaje informacji o jednym obiekcie, czyli o obcym w grze. Jednak słownik można wykorzystać także do przechowywania jednego rodzaju informacji o wielu obiektach. Na przykład przeprowadzamy ankietę dotyczącą ulubionego języka programowania. W takim przypadku słownik będzie użyteczną strukturą przeznaczoną do przechowywania wyników tak prostej ankiety:

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }
```

Jak możesz zobaczyć, definicja większego słownika została podzielona na kilka wierszy kodu. Każdy klucz to imię uczestnika ankiety, natomiast wartość to podany przez niego ulubiony język programowania. Kiedy wiesz, że do utworzenia słownika potrzebujesz więcej niż tylko jednego wiersza kodu, po nawiasie otwierającym naciśnij klawisz *Enter*. W ten sposób powstanie wcięcie o wielkości jednego poziomu (cztery spacje) i zostanie zapisana pierwsza para klucz-wartość wraz z przecinkiem. Od tego momentu kolejne naciśnięcia klawisza *Enter* powinny powodować, że edytor tekstu automatycznie będzie stosował wcięcia dla następnych par klucz-wartość, aby dopasować wielkość tych wcięć do pierwszej pary.

Gdy zakończysz definiowanie słownika, w nowym wierszu po ostatniej parze klucz-wartość umieść nawias zamykający i zastosuj wcięcie na poziomie równym kluczom słownika. Dobrą praktyką jest umieszczanie przecinka także po ostatniej parze klucz-wartość, aby definicja słownika była gotowa na dodanie nowej pary klucz-wartość w następnym wierszu.

UWAGA *Większość edytorów oferuje pewnego rodzaju funkcjonalność pomagającą w formatowaniu rozbudowanych list i słowników w sposób podobny do przedstawionego w przykładzie. Dostępne są jeszcze inne akceptowalne sposoby formatowania długich słowników, więc w używanym edytorze oraz w innych źródłach będziesz mógł się spotkać z nieco odmiennym formatowaniem.*

Dzięki tak przygotowanemu słownikowi, znając imię uczestnika ankiety, możemy bardzo łatwo pobrać informacje o jego ulubionym języku programowania.

Plik favorite_languages.py*:*

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }

language = favorite_languages['sara'].title() ❶
print(f"Ulubiony język programowania Sary to {language}.")
```

W celu wyświetlenia ulubionego języka programowania Sary należy użyć przedstawionego poniżej polecenia:

```
favorite_languages['sara']
```

Powyższa składnia została użyta do pobrania ze słownika ulubionego języka programowania Sary ❶ i przypisania go zmiennej `language`. Dzięki utworzeniu nowej zmiennej otrzymujemy znacznie czytelniejsze wywołanie `print()`. Wygenerowane dane wyjściowe zawierają informacje o ulubionym języku programowania Sary, tak jak pokazałem poniżej:

```
Ulubiony język programowania Sary to C.
```

Tę samą składnię można wykorzystać do przedstawienia dowolnego elementu ze słownika.

Używanie metody get() w celu uzyskania dostępu do wartości

Używanie umieszczonych w nawiasach kwadratowych kluczy w celu pobierania żądanych wartości ze słownika może prowadzić do potencjalnego problemu: jeśli podany klucz nie istnieje, wynikiem będzie błąd.

Zobacz, co się stanie, gdy spróbujesz pobrać ze słownika nieistniejącą wartość:

Plik alien_no_points.py*:*

```
alien_0 = {'color': 'zielony', 'speed': 'wolno'}
print(alien_0['points'])
```

Wynikiem jest wyświetlenie stosu wywołań wskazującego na wystąpienie błędu typu `KeyError`:

```
Traceback (most recent call last):
  File "alien_no_points.py", line 2, in <module>
    print(alien_0['points'])
KeyError: 'points'
```

W rozdziale 10. znajdziesz więcej informacji na temat obsługi błędów takich jak w omawianym przykładzie. Podczas pracy ze słownikiem można wykorzystać metodę `get()` do zdefiniowania wartości domyślnej, która będzie zwrócona, jeśli żądany klucz nie istnieje.

Metoda `get()` wymaga podania klucza jako pierwszego argumentu. Drugim, opcjonalnym argumentem jest wartość zwracana w przypadku, gdy podany klucz nie istnieje w słowniku.

```
alien_0 = {'color': 'zielony', 'speed': 'wolno'}

point_value = alien_0.get('points', 'Brak przypisanych punktów.')
print(point_value)
```

Jeżeli klucz `'points'` istnieje w słowniku, otrzymasz przypisaną mu wartość. Natomiast jeśli klucz nie istnieje, otrzymasz wartość domyślną. W omawianym przykładzie klucz `'points'` nie istnieje w słowniku, więc zamiast błędu otrzymujesz czytelny komunikat:

```
Brak przypisanych punktów.
```

Jeżeli istnieje niebezpieczeństwo, że żądany klucz nie znajduje się w słowniku, rozważ wykorzystanie metody `get()` zamiast składni z użyciem nawiasu kwadratowego.

UWAGA *Jeżeli pominiesz drugi argument w wywołaniu `get()`, a klucz nie istnieje, wówczas Python zwróci wartość `None`. Jest to wartość specjalna oznaczająca „brak wartości". To nie jest błąd, lecz jedynie wartość specjalna wskazująca na brak wyraźnie zdefiniowanej wartości. Więcej przykładów użycia `None` przedstawię w rozdziale 8.*

ZRÓB TO SAM

6.1. Osoba. Wykorzystaj słownik do przechowywania informacji o znanej Ci osobie. W słowniku powinny znaleźć się informacje takie jak imię, nazwisko, wiek i miasto zamieszkania. Powinieneś więc utworzyć klucze `first_name`, `last_name`, `age` i `city`. Następnie wyświetl wszystkie informacje przechowywane w słowniku.

6.2. Ulubione liczby. Wykorzystaj słownik do przechowywania ulubionych liczb różnych osób. Weź pod uwagę pięć osób i ich imion użyj w charakterze kluczy słownika. Następnie ustal ich ulubione liczby i umieść je w słowniku, przypisując każdej osobie po jednej liczbie. Wyświetl imiona wszystkich osób i ich ulubione liczby. Jeżeli chcesz mieć więcej frajdy podczas wykonywania tego ćwiczenia, zapytaj przyjaciół o ich ulubione liczby i umieść w programie rzeczywiste dane.

6.3. Glosariusz. Słownik Pythona można wykorzystać do przygotowania rzeczywistego słownika. Jednak w celu uniknięcia niejasności nazwiemy go glosariuszem.

- Wypisz sobie pięć słów z dziedziny programowania, które poznałeś we wcześniejszych rozdziałach. Te słowa będą kluczami w glosariuszu, natomiast wartościami będą znaczenia poszczególnych słów.
- Każde słowo i jego znaczenie wyświetl w postaci elegancko sformatowanych danych wyjściowych. Możesz w jednym wierszu wyświetlić słowo, dwukropek i później wyjaśnienie danego słowa. Ewentualnie słowo umieść w jednym wierszu, a jego wyjaśnienie w następnym, wciętym wierszu. Do wstawienia pustego wiersza między parami słowo-definicja użyj znaku nowego wiersza (`\n`).

Iteracja przez słownik

Pojedynczy słownik Pythona może zawierać od kilku do nawet kilku milionów par klucz-wartość. Ponieważ w słowniku może znaleźć się ogromna ilość danych, Python pozwala przeprowadzać iterację przez słownik. Ponadto słowniki mogą być wykorzystywane do przechowywania informacji na wiele różnych sposobów, więc istnieje kilka odmiennych rozwiązań w zakresie iteracji przez słownik. Możliwa jest iteracja przez wszystkie pary klucz-wartość słownika albo tylko przez jego klucze lub wartości.

Iteracja przez wszystkie pary klucz-wartość

Zanim przejdziemy do omawiania różnych podejść w zakresie iteracji, najpierw spójrz na nowy słownik przeznaczony do przechowywania informacji o użytkowniku witryny internetowej. Przedstawiony poniżej słownik zawiera nazwę użytkownika, imię oraz nazwisko jednej osoby:

```
user_0 = {
    'username': 'jkowalski',
    'first': 'jan',
    'last': 'kowalski',
    }
```

Na podstawie wiedzy zdobytej dotąd w tym rozdziale potrafisz uzyskać dostęp do pojedynczej informacji dotyczącej użytkownika, którego dane znajdują się w słowniku `user_0`. Co możesz zrobić w sytuacji, gdy ze słownika chcesz pobrać wszystkie informacje o danym użytkowniku? W tym celu za pomocą pętli `for` możesz przeprowadzić iterację przez słownik.

Plik user.py*:*

```
user_0 = {
    'username': 'jkowalski',
    'first': 'jan',
    'last': 'kowalski',
    }

for key, value in user_0.items(): ❶
    print(f"\nKlucz: {key}") ❷
    print(f"Wartość: {value}") ❸
```

Jak pokazałem w wierszu ❶, w celu przygotowania pętli `for` dla słownika konieczne jest utworzenie dwóch zmiennych przechowujących klucz i wartość każdej pary klucz-wartość. Dla wspomnianych zmiennych możesz wybrać dowolne nazwy. Powyższy kod będzie również działał bez problemów, jeśli dla nazw zmiennych użyjesz skrótów, na przykład:

```
for k, v in user_0.items()
```

W części drugiej polecenia zdefiniowanego w wierszu ❶ mamy nazwę słownika oraz metodę `items()`, której wartością zwrotną jest lista par klucz-wartość. Następnie pętla `for` przechowuje poszczególne pary w dwóch przedstawionych zmiennych. W omawianym fragmencie kodu zmienne wykorzystujemy do wyświetlenia klucza (patrz wiersz ❷) oraz przypisanej mu wartości (patrz wiersz ❸). Sekwencja `\n` w pierwszym poleceniu `print()` zapewnia wstawienie w wygenerowanych danych wyjściowych pustego wiersza przed każdą parą klucz-wartość:

```
Klucz: username
Wartość: jkowalski

Klucz: first
Wartość: jan

Klucz: last
Wartość: kowalski
```

Iteracja przez wszystkie pary klucz-wartość sprawdza się wyjątkowo dobrze w przypadku słowników takich jak ten użyty w przedstawionym wcześniej programie *favorite_languages.py*, który przechowuje ten sam rodzaj informacji dla wielu różnych kluczy. Jeżeli przeprowadzisz iterację przez słownik `favorite_languages`, dane wyjściowe będą zawierały imię każdej osoby w słowniku oraz jej ulubiony język programowania. Ponieważ klucze zawsze odwołują się do imienia osoby, a wartość zawsze przedstawia język programowania, więc zmiennym w pętli `for` można nadać nazwy `name` i `language` zamiast ogólnych `key` i `value`. To znacznie ułatwi zrozumienie przeznaczenia danej pętli.

Plik favorite_languages.py*:*

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }

for name, language in favorite_languages.items(): ❶
    print(f"Ulubiony język programowania użytkownika {name.title()}
    ↪to {language.title()}.") ❷
```

W wierszu ❶ nakazujemy Pythonowi przeprowadzenie iteracji przez wszystkie pary klucz-wartość w słowniku. Podczas tej operacji klucz każdej pary jest przechowywany w zmiennej `name`, natomiast jego wartość w zmiennej `language`. Tego rodzaju jasne i czytelne nazwy znacznie ułatwiają pokazanie, na czym polega działanie wywołania `print()` w wierszu ❷.

W ten sposób za pomocą zaledwie kilku wierszy kodu można wyświetlić wszystkie informacje otrzymane od uczestników ankiety:

```
Ulubiony język programowania użytkownika Janek to Python.
Ulubiony język programowania użytkownika Sara to C.
Ulubiony język programowania użytkownika Paweł to Python.
Ulubiony język programowania użytkownika Edward to Ruby.
```

Taki rodzaj pętli będzie się sprawdzał równie dobrze, gdy słownik będzie zawierał wyniki ankiety, w której wzięło udział na przykład milion osób.

Iteracja przez wszystkie klucze słownika

Metoda `keys()` jest użyteczna, gdy nie trzeba przetwarzać wszystkich wartości znajdujących się w słowniku. W poniższym fragmencie kodu przeprowadzamy iterację przez słownik `favorite_languages` i wyświetlamy imiona wszystkich uczestników ankiety:

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }

for name in favorite_languages.keys(): ❶
    print(name.title())
```

W wierszu ❶ nakazujemy Pythonowi pobranie wszystkich kluczy ze słownika `favorite_languages` i przechowujemy je pojedynczo w zmiennej `name`. Wygenerowane dane wyjściowe zawierają imiona wszystkich uczestników ankiety:

```
Janek
Sara
Paweł
Edward
```

Iteracja przez klucze to tak naprawdę zachowanie domyślne podczas iteracji przez słownik, więc te same dane wyjściowe otrzymasz też po użyciu polecenia:

```
for name in favorite_languages:
```

zamiast:

```
for name in favorite_languages.keys():
```

Możesz zdecydować się na wyraźne użycie metody `keys()`, jeśli dzięki temu kod będzie łatwiejszy do odczytania, lub zupełnie ją pominąć.

Wewnątrz pętli `for` możesz uzyskać dostęp do wartości powiązanej z kluczem, co wymaga użycia bieżącego klucza. Kilku przyjaciołom wyświetlimy teraz komunikat o wybranych przez nich językach programowania. Podobnie jak to robiliśmy wcześniej, przeprowadzamy iterację przez imiona w słowniku, ale kiedy dopasujemy imię do jednego z naszych przyjaciół, wyświetlamy komunikat dotyczący jego ulubionego języka programowania:

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }

friends = ['paweł', 'sara'] ❶
for name in favorite_languages.keys():
    print(f"Witaj, {name.title()}.")

    if name in friends: ❷
        language = favorite_languages[name].title() ❸
        print(f"\tWitaj, {name.title()}! Widzę, że Twoim ulubionym
            językiem programowania jest {language}!")
```

W wierszu ❶ tworzymy listę przyjaciół, którym chcemy wyświetlić komunikat. Wewnątrz pętli wyświetlamy imiona wszystkich osób. Następnie w wierszu ❷ sprawdzamy, czy imię aktualnie przechowywane w zmiennej `name` odpowiada imieniu znajdującemu się na liście `friends`. Jeżeli tak, dostęp do ulubionego języka programowania, używamy nazwy słownika oraz bieżącej wartości `name` jako klucza (patrz wiersz ❸). Następnie wyświetlamy specjalne powitanie odwołujące się do ulubionego języka programowania.

Wygenerowane dane wyjściowe zawierają imiona wszystkich osób, natomiast nasi przyjaciele otrzymują jeszcze specjalny komunikat:

```
Edward
Paweł
  Witaj, Paweł! Widzę, że Twoim ulubionym językiem programowania jest
  ↪Python!
Sara
  Witaj, Sara! Widzę, że Twoim ulubionym językiem programowania jest C!
Janek
```

Możliwe jest również użycie metody `keys()` do odszukania określonej osoby, która wzięła udział w ankiecie. Tym razem sprawdzamy, czy Elżbieta wzięła udział w ankiecie:

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }

if 'elżbieta' not in favorite_languages.keys(): ❶
    print("Elżbieta, proszę, weź udział w naszej ankiecie!")
```

Metoda `keys()` nie służy jedynie do przeprowadzania iteracji. W rzeczywistości zwraca listę wszystkich kluczy, a kod przedstawiony w wierszu ❶ po prostu sprawdza, czy `'elżbieta'` znajduje się na liście. Ponieważ Elżbiety nie ma na liście, zachęcamy ją do wzięcia udziału w ankiecie:

```
Elżbieta, proszę, weź udział w naszej ankiecie!
```

Iteracja przez uporządkowane klucze słownika

Począwszy od wydania Python 3.7, iteracja przez słownik zawsze zwraca elementy w kolejności ich wstawiania do słownika. Jednak czasami zachodzi potrzeba przeprowadzenia iteracji w zupełnie innej kolejności.

Jedynym sposobem, aby elementy zostały zwrócone w określonej kolejność, jest posortowanie kluczy po ich otrzymaniu w pętli `for`. Funkcję `sorted()` możemy wykorzystać do uzyskania uporządkowanych kopii kluczy:

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }

for name in sorted(favorite_languages.keys()):
    print(f"{name.title()}, dziękujemy za udział w ankiecie.")
```

Powyższe polecenie `for` jest podobne do wcześniejszych, z wyjątkiem tego, że wywołanie metody `dictionary.keys()` zostało opakowane funkcją `sorted()`. W ten sposób nakazaliśmy Pythonowi wyświetlenie wszystkich kluczy słownika oraz posortowanie listy przed przeprowadzeniem iteracji. Wygenerowane dane wyjściowe pokazują, że imiona uczestników ankiety zostały wyświetlone w kolejności alfabetycznej:

```
Edward, dziękujemy za udział w ankiecie.
Janek, dziękujemy za udział w ankiecie.
Paweł, dziękujemy za udział w ankiecie.
Sara, dziękujemy za udział w ankiecie.
```

Iteracja przez wszystkie wartości słownika

Jeżeli interesują nas przede wszystkim wartości przechowywane w słowniku, wówczas można wykorzystać metodę `values()` w celu zwrotu listy wartości bez jakichkolwiek kluczy. Przykładowo przyjmujemy założenie, że chcemy pobrać listę wszystkich języków programowania wymienionych przez uczestników ankiety i nie interesują nas imiona osób wskazujących poszczególne języki:

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
    'paweł': 'python',
    }

print("W ankiecie zostały wymienione następujące języki programowania:")
for language in favorite_languages.values():
    print(language.title())
```

Powyższe pętla `for` pobiera wszystkie wartości ze słownika i przechowuje je w zmiennej `language`. Po wyświetleniu poszczególnych wartości otrzymujemy listę wszystkich języków programowania wymienionych przez uczestników ankiety:

```
W ankiecie zostały wymienione następujące języki programowania:
Python
C
Ruby
Python
```

Tego rodzaju podejście pobiera wszystkie wartości ze słownika bez sprawdzania, czy się powtarzają. Przedstawione rozwiązanie może być wystarczające dla małej liczby wartości, ale w przypadku ankiety przeprowadzanej na ogromnej liczbie respondentów otrzymamy listę z wieloma powtarzającymi się wartościami. Aby wyświetlić jedynie unikatowe wartości, możemy użyć zbioru. Wspomniany *zbiór* jest podobny do listy, ale każdy znajdujący się w nim element musi być unikatowy:

```
favorite_languages = {
    'janek': 'python',
    'sara': 'c',
    'edward': 'ruby',
```

```
    'paweł': 'python',
    }

print("W ankiecie zostały wymienione następujące języki programowania:")
for language in set(favorite_languages.values()): ❶
    print(language.title())
```

Kiedy lista zawierająca powielające się elementy jest opakowana wywołaniem `set()`, Python identyfikuje wszystkie unikatowe elementy na liście, a następnie na ich podstawie tworzy zbiór. W wierszu ❶ wykorzystujemy wywołanie `set()` do pobrania unikatowych nazw języków otrzymanych jako wynik działania metody `favorite_language.values()`.

Wynikiem jest lista języków wymienionych przez uczestników ankiety, która nie zawiera powtarzających się elementów:

```
W ankiecie zostały wymienione następujące języki programowania:
Python
C
Ruby
```

Gdy będziesz kontynuować poznawanie Pythona, bardzo często odkryjesz wbudowane funkcje języka pomagające w przetworzeniu danych dokładnie w oczekiwany przez Ciebie sposób.

UWAGA *Zbiór można utworzyć bezpośrednio za pomocą nawiasu klamrowego, w którym trzeba umieścić rozdzielone przecinkami elementy:*

```
>>> languages = {'python', 'ruby', 'python', 'c'}
>>> languages
{'ruby', 'python', 'c'}
```

Bardzo łatwo pomylić zbiór ze słownikiem, ponieważ w obu przypadkach stosowany jest nawias klamrowy. Gdy widzisz nawias klamrowy, ale bez par klucz-wartość, prawdopodobnie masz do czynienia ze zbiorem. W przeciwieństwie do list i słowników zbiór nie przechowuje elementów w żadnej konkretnej kolejności.

ZRÓB TO SAM

6.4. Glosariusz 2. Skoro już wiesz, jak można przeprowadzić iterację przez słownik, to zmodyfikuj kod ćwiczenia 6.3 z wcześniejszej części rozdziału. Zastąp serię wywołań `print()` pętlą przeprowadzającą iterację przez klucze i wartości słownika. Po upewnieniu się, że pętla działa prawidłowo, do glosariusza dodaj kolejnych pięć terminów związanych z Pythonem. Kiedy ponownie uruchomisz program, nowo dodane terminy i ich definicje powinny zostać automatycznie uwzględnione w wyświetlonych danych wyjściowych.

6.5. Rzeki. Utwórz słownik zawierający trzy ważne rzeki oraz kraje, przez które one płyną. Jedna z par klucz-wartość może mieć postać `'nil': 'egipt'`.

- Wykorzystaj pętlę do wyświetlenia zdania o każdej rzece, na przykład: „Nil przepływa przez Egipt".
- Wykorzystaj pętlę do wyświetlenia nazw wszystkich rzek przechowywanych w słowniku.
- Wykorzystaj pętlę do wyświetlenia nazw wszystkich państw przechowywanych w słowniku.

6.6. Ankieta. Użyj kodu znajdującego się w programie *favorite_languages.py*, utworzonym nieco wcześniej w tym rozdziale.

- Utwórz listę osób, które powinny wziąć udział w ankiecie dotyczącej ulubionego języka programowania. Umieść na niej pewne osoby, które już znajdują się w słowniku, oraz te, które jeszcze nie zostały zapisane w słowniku.
- Przeprowadź iterację przez listę osób, które powinny wziąć udział w ankiecie. Jeżeli dana osoba już wzięła udział w ankiecie, wyświetl komunikat z podziękowaniem za jej zaangażowanie. Natomiast jeśli dana osoba jeszcze nie udzieliła odpowiedzi w ankiecie, wyświetl komunikat z zaproszeniem do wzięcia w niej udziału.

Zagnieżdżanie

Czasami zachodzi potrzeba przechowywania zestawu słowników na liście lub listy elementów jako wartości słownika. Mamy wówczas do czynienia z *zagnieżdżaniem*. Istnieje możliwość zagnieżdżenia zestawu słowników na liście, listy elementów wewnątrz słownika lub nawet słownika wewnątrz innego słownika. Zagnieżdżanie to funkcja o potężnych możliwościach, o czym się przekonasz, analizując następne przykłady.

Lista słowników

Słownik `alien_0` zawiera wiele różnych informacji o jednym obcym, ale nie ma już miejsca do przechowywania informacji o drugim obcym, nie wspominając już o ekranie pełnym obcych. W jaki sposób można zarządzać flotą obcych? Jednym z rozwiązań jest utworzenie listy obcych, na której każdy obcy będzie przedstawiony za pomocą słownika zawierającego informacje o nim. Na przykład w przedstawionym poniżej kodzie mamy listę dotyczącą trzech obcych.

Plik aliens.py*:*

```
alien_0 = {'color': 'zielony', 'points': 5}
alien_1 = {'color': 'żółty', 'points': 10}
alien_2 = {'color': 'czerwony', 'points': 15}
```

```
aliens = [alien_0, alien_1, alien_2] ❶

for alien in aliens:
    print(alien)
```

Najpierw tworzymy trzy słowniki, z których każdy przedstawia innego obcego. Polecenie w wierszu ❶ umieszcza wszystkie słowniki na liście o nazwie `aliens`. Na końcu przeprowadzamy iterację przez listę i wyświetlamy informacje o każdym obcym:

```
{'color': 'zielony', 'points': 5}
{'color': 'żółty', 'points': 10}
{'color': 'czerwony', 'points': 15}
```

Znacznie bardziej rzeczywisty przykład dotyczy utworzenia więcej niż tylko trzech obcych za pomocą kodu, który będzie ich automatycznie generował. Spójrz na poniższy fragmentu kodu, w którym wykorzystujemy funkcję `range()` do przygotowania floty 30 obcych:

```
# Utworzenie pustej listy przeznaczonej do przechowywania obcych.
aliens = []

# Utworzenie 30 zielonych obcych.
for alien_number in range(30): ❶
    new_alien = {'color': 'zielony', 'points': 5, 'speed': 'wolno'} ❷
    aliens.append(new_alien) ❸

# Wyświetlenie pierwszych pięciu obcych.
for alien in aliens[:5]: ❹
    print(alien)
print("...")

# Wyświetlenie całkowitej liczby utworzonych obcych.
print(f"Całkowita liczba obcych: {len(aliens)}") ❺
```

Ten przykład rozpoczyna się od pustej listy przeznaczonej do przechowywania wszystkich obcych, którzy zostaną utworzeni. W wierszu ❶ wynikiem działania funkcji `range()` jest zbiór liczb wskazujący Pythonowi, ile razy ma być powtórzona pętla. W trakcie każdej iteracji pętli tworzymy nowego obcego (patrz wiersz ❷), a następnie dołączamy go do listy `aliens` (patrz wiersz ❸). Z kolei w wierszu ❹ używamy wycinka do wyświetlenia pierwszych pięciu obcych. Na końcu (patrz wiersz ❺) wyświetlamy wielkość listy, co potwierdza wygenerowanie pełnej floty 30 obcych:

```
{'color': 'zielony', 'points': 5, 'speed': 'wolno'}
{'color': 'zielony', 'points': 5, 'speed': 'wolno'}
{'color': 'zielony', 'points': 5, 'speed': 'wolno'}
{'color': 'zielony', 'points': 5, 'speed': 'wolno'}
{'color': 'zielony', 'points': 5, 'speed': 'wolno'}
...

Całkowita liczba obcych: 30
```

Wprawdzie każdy wygenerowany obcy ma tę samą charakterystykę, ale każdego z nich Python uznaje za oddzielny obiekt, co pozwala nam na modyfikację poszczególnych obcych.

Jak można pracować z tego rodzaju zbiorem obcych? Wyobraź sobie, że jednym z aspektów gry jest zmiana koloru obcego i jego szybkości wraz z postępem poczynionym przez gracza. Kiedy nadchodzi czas zmiany koloru, można wykorzystać pętlę `for` i polecenie `if` do zmiany koloru obcego. Na przykład aby zmienić kolor pierwszych trzech obcych na żółty, ich szybkość na średnią, a wartość na 10 punktów, możesz skorzystać z przedstawionego poniżej kodu:

```
# Utworzenie pustej listy przeznaczonej do przechowywania obcych.
aliens = []

# Utworzenie 30 zielonych obcych.
for alien_number in range (30):
    new_alien = {'color': 'zielony', 'points': 5, 'speed': 'wolno'}
    aliens.append(new_alien)

for alien in aliens[0:3]:
    if alien['color'] == 'zielony':
        alien['color'] = 'żółty'
        alien['speed'] = 'średnio'
        alien['points'] = 10

# Wyświetlenie pierwszych pięciu obcych:
for alien in aliens[:5]:
    print(alien)
print("...")
```

Ponieważ chcemy zmodyfikować jedynie trzech pierwszych obcych, przeprowadzamy iterację przez wycinek zawierający trzy pierwsze elementy listy `aliens`. Obecnie wszyscy obcy są koloru zielonego, ale przecież nie zawsze tak będzie. Dlatego też w kodzie umieszczamy polecenie `if` dające pewność, że zmodyfikujemy jedynie zielonych obcych. Jeżeli obcy ma kolor zielony, zmieniamy go na żółty, a szybkość poruszania się obcego na średnią. Ponadto po zestrzeleniu takiego obcego gracz otrzyma 10 punktów, jak to wynika z poniższych danych wyjściowych:

```
{'color': 'żółty', 'points': 10, 'speed': 'średnio'}
{'color': 'żółty', 'points': 10, 'speed': 'średnio'}
{'color': 'żółty', 'points': 10, 'speed': 'średnio'}
{'color': 'zielony', 'points': 5, 'speed': 'wolno'}
{'color': 'zielony', 'points': 5, 'speed': 'wolno'}
...
```

Tę pętlę można jeszcze bardziej rozbudować przez dodanie bloku `elif` zmieniającego żółtego obcego w czerwonego, który porusza się szybko i jest wart 15 punktów. Poniżej przedstawiam jedynie fragment programu, w którym wprowadzamy tę zmianę:

```
for alien in aliens[0:3]:
    if alien['color'] == 'zielony':
        alien['color'] = 'żółty'
        alien['speed'] = 'średnio'
        alien['points'] = 10
    elif alien['color'] == 'żółty':
        alien['color'] = 'czerwony'
        alien['speed'] = 'szybko'
        alien['points'] = 15
```

Bardzo często zdarza się przechowywać pewną liczbę słowników na liście, gdy każdy z tych słowników zawiera wiele rodzajów informacji dotyczących jednego obiektu. Przykładowo można utworzyć słownik dla każdego użytkownika witryny internetowej, tak jak w programie *user.py* przedstawionym wcześniej w tym rozdziale, a następnie przechowywać poszczególne słowniki na liście o nazwie `users`. Wszystkie słowniki na liście powinny mieć identyczną strukturę, aby można było przeprowadzić iterację listy i pracować z każdym obiektem słownika w taki sam sposób.

Lista w słowniku

Zamiast umieszczać słownik na liście, czasami użyteczne będzie umieszczenie listy w słowniku. Zastanówmy się na przykład nad sposobem przedstawienia pizzy zamawianej przez klienta. Jeżeli do dyspozycji mamy jedynie listę, wówczas tak naprawdę możemy przechowywać tylko dodatki wybrane przez klienta. Natomiast w przypadku słownika lista dodatków będzie jednym z aspektów pizzy, o których informacje możemy przechowywać.

W poniższym fragmencie kodu dla każdej pizzy przechowujemy dwa rodzaje informacji: grubość ciasta oraz listę dodatków. Wspomniana lista dodatków to wartość przypisana kluczowi `'toppings'`. Aby użyć elementu z tej listy, należy podać nazwę słownika i klucza `'toppings'`, podobnie jak to się robi w przypadku dowolnej wartości słownika. Zamiast pojedynczej wartości otrzymujemy listę dodatków.

Plik pizza.py*:*

```
# Przechowywanie informacji o pizzy zamawianej przez klienta.
pizza = { ❶
    'crust': 'grubym',
    'toppings': ['pieczarki', 'podwójny ser'],
    }

# Podsumowanie zamówienia.
print(f"Zamówiłeś pizzę na {pizza['crust']} cieście " ❷
    "wraz z następującymi dodatkami:")

for topping in pizza['toppings']: ❸
    print(f"\t{topping}")
```

Rozpoczynamy od utworzenia w wierszu ❶ słownika przeznaczonego na przechowywanie informacji o zamawianej pizzy. Jednym z kluczy słownika jest `'crust'`, któremu przypisaliśmy wartość w postaci ciągu tekstowego `'grubym'`. Kolejny klucz `'toppings'` ma wartość w postaci listy przechowującej wszystkie dodatki wybrane przez klienta. W wierszu ❷ generujemy podsumowanie zamówienia, zanim rozpoczniemy przygotowywanie pizzy. Gdy zachodzi potrzeba podziału długiego wiersza w wywołaniu `print()`, wybierz odpowiedni punkt podziału wiersza i zakończ go znakiem cytowania. Przejdź do następnego wiersza, dodaj wcięcie, dodaj otwierający znak cytowania i kontynuuj ciąg tekstowy. Python automatycznie połączy wszystkie ciągi tekstowe znalezione w nawiasie wywołania `print()`. W celu wyświetlenia dodatków tworzymy pętlę `for` (patrz wiersz ❸). Aby uzyskać dostęp do listy dodatków, używamy klucza `'toppings'`, a Python pobiera listę dodatków ze słownika.

Poniższe dane wyjściowe wskazują, jaka powinna być pizza, którą zamierzamy przygotować:

```
Zamówiłeś pizzę na grubym cieście wraz z następującymi dodatkami:
    pieczarki
    podwójny ser
```

Istnieje możliwość zagnieżdżenia listy wewnątrz słownika za każdym razem, gdy zachodzi konieczność przypisania więcej niż tylko jednej wartości pojedynczemu kluczowi w słowniku. W omawianym wcześniej przykładzie z ulubionym językiem programowania, gdybyśmy mieli możliwość przechowywania odpowiedzi respondenta na liście, każdy z nich mógłby wskazać więcej niż tylko jeden ulubiony język programowania. Podczas iteracji przez słownik program przypisałby poszczególnym osobom jako wartość listę języków, a nie tylko jeden język. Wewnątrz pętli `for` słownika używamy więc następnej pętli `for`, tym razem do przeprowadzenia iteracji listy języków podanych przez poszczególne osoby.

Plik favorite_languages.py*:*

```
favorite_languages = { ❶
    'janek': ['python', 'ruby'],
    'sara': ['c'],
    'edward': ['ruby', 'go'],
    'paweł': ['python', 'haskell'],
    }

for name, languages in favorite_languages.items(): ❷
    print(f"\nUlubione języki programowania użytkownika {name.title()} to:")
    for language in languages: ❸
        print(f"\t{language.title()}")
```

Jak możesz zobaczyć w wierszu ❶, wartością przypisywaną poszczególnym osobom jest teraz lista. Zwróć uwagę na to, że część osób podaje tylko jeden ulubiony język programowania, podczas gdy inne kilka. W trakcie iteracji przez słownik (patrz wiersz ❷) zmiennej o nazwie `languages` używamy do przechowywania każdej wartości ze słownika, ponieważ teraz wiemy, że będzie listą. Wewnątrz głównej pętli słownika wykorzystujemy inną pętlę `for` (patrz wiersz ❸) do przeprowadzenia iteracji przez listę ulubionych języków każdej osoby. W tym momencie respondent może podać dowolną liczbę ulubionych języków programowania:

```
Ulubione języki programowania użytkownika Janek to:
    Python
    Ruby

Ulubione języki programowania użytkownika Sara to:
    C

Ulubione języki programowania użytkownika Edward to:
    Ruby
    Go

Ulubione języki programowania użytkownika Paweł to:
    Python
    Haskell
```

Aby jeszcze bardziej dopracować program, możemy na początku pętli `for` przeprowadzającej iterację przez słownik dodać polecenie `if` sprawdzające, czy dana osoba wskazała więcej niż tylko jeden ulubiony język programowania. Wspomniane sprawdzenie odbywa się przez analizę wartości wyniku wywołania `len(languages)`. Jeżeli osoba podała więcej niż tylko jeden ulubiony język programowania, dane wyjściowe pozostaną takie same. Natomiast w przypadku podania tylko jednego ulubionego języka, zmienimy treść komunikatu, na przykład na następujący: „Ulubiony język programowania użytkownika Sara to C”.

UWAGA *Nie powinieneś za bardzo zagnieżdżać list i słowników. Jeżeli zagnieżdżasz elementy w większym stopniu, niż pokazałem we wcześniejszych przykładach, lub pracujesz z utworzonym przez innych kodem ze znacznym poziomem zagnieżdżenia, prawdopodobnie oznacza to, że istnieje prostszy sposób rozwiązania danego problemu.*

Słownik w słowniku

Można zagnieżdżać słownik w innym słowniku, ale w takim przypadku kod bardzo szybko staje się dość skomplikowany. Na przykład jeśli z witryny internetowej będzie korzystało wielu użytkowników o unikatowych nazwach, nazwy tych użytkowników będzie można wykorzystać w charakterze kluczy słownika. Wówczas informacje o poszczególnych użytkownikach mogą być przechowywane przy użyciu słownika jako wartości przypisanej do nazwy użytkownika. W poniższym programie przechowujemy trzy rodzaje informacji o każdym użytkowniku: imię, nazwisko i miejscowość. Dostęp do tych informacji odbywa się za pomocą iteracji przez nazwy użytkowników i powiązane z nimi słowniki z informacjami.

Plik many_users.py*:*

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
        },
    'mcurie': {
        'first': 'maria',
        'last': 'skłodowska-curie',
        'location': 'paryż',
        },
    }

for username, user_info in users.items(): ❶
    print(f"\nNazwa użytkownika: {username}") ❷
    full_name = f"{user_info['first']} {user_info['last']}" ❸
    location = user_info['location']

    print(f"\tImię i nazwisko: {full_name.title()}") ❹
    print(f"\tMiejscowość: {location.title()}")
```

Zaczynamy od zdefiniowania słownika o nazwie `users` wraz z dwoma kluczami, po jednym dla nazw użytkowników `'aeinstein'` i `'mcurie'`. Wartością powiązaną z każdym kluczem będzie słownik zawierający imię, nazwisko oraz miejscowość. W wierszu ❶ przeprowadzamy iterację przez słownik `users`. Python przechowuje każdy klucz w zmiennej `username`, natomiast powiązany z nim słownik zostaje umieszczony w zmiennej `user_info`. Kod w wierszu ❷ pętli głównej powoduje wyświetlenie nazwy użytkownika.

W wierszu ❸ rozpoczyna się uzyskiwanie dostępu do wewnętrznego słownika. Zmienna `user_info` zawierająca słownik informacji o użytkowniku ma trzy klucze: `'first'`, `'last'` i `'location'`. Poszczególne klucze wykorzystujemy do wygenerowania elegancko sformatowanego pełnego imienia i nazwiska oraz miejscowości danej osoby. Następnie wyświetlamy podsumowanie informacji o tej osobie (patrz wiersz ❹):

```
Nazwa użytkownika: aeinstein
    Imię i nazwisko: Albert Einstein
    Miejscowość: Princeton

Nazwa użytkownika: mcurie
    Imię i nazwisko: Maria Skłodowska-Curie
    Miejscowość: Paryż
```

Zwróć uwagę, że struktura słowników utworzonych dla poszczególnych użytkowników jest identyczna. Wprawdzie to nie jest wymagane przez Pythona, ale dzięki wykorzystywaniu takiej samej struktury łatwiej jest pracować z zagnieżdżonymi słownikami. Jeżeli słowniki utworzone dla poszczególnych użytkowników miałyby inne klucze, wtedy kod wewnątrz pętli `for` byłby znacznie bardziej skomplikowany.

ZRÓB TO SAM

6.7. Osoby. Pracę rozpocznij od programu stworzonego w ćwiczeniu 6.1 we wcześniejszej części rozdziału. Utwórz dwa nowe słowniki przedstawiające różne osoby, a następnie wszystkie trzy słowniki umieść na liście o nazwie `people`. Przeprowadź iterację przez listę osób i wyświetl wszystkie informacje o poszczególnych osobach.

6.8. Zwierzęta. Utwórz kilka słowników i nadaj im nazwy zwierząt. W poszczególnych słownikach umieść informacje o zwierzętach będących ich właścicielami. Następnie te słowniki powinny znaleźć się na liście o nazwie `pets`. Teraz przeprowadź iterację przez listę i wyświetl wszystkie informacje o poszczególnych zwierzętach.

6.9. Ulubione miejsca. Utwórz słownik o nazwie `favorite_places`. Pomyśl o trzech imionach i użyj ich jako kluczy słownika. Każdej osobie przypisz po trzy ulubione miejsca. Aby ćwiczenie stało się jeszcze bardziej interesujące, możesz poprosić przyjaciół o podanie ulubionych miejsc. Przeprowadź iterację przez słownik i wyświetl imiona wszystkich osób oraz ich ulubione miejsca.

6.10. Ulubione liczby. Zmodyfikuj program utworzony w ćwiczeniu 6.2 we wcześniejszej części rozdziału. Po zmianach każda osoba może mieć więcej niż tylko jedną ulubioną liczbę. Wyświetl wszystkie osoby oraz ich ulubione liczby.

6.11. Miasta. Utwórz słownik o nazwie `cities`. Jako klucze podaj nazwy trzech miast. Dla każdego z nich utwórz oddzielny słownik zawierający informacje o danym mieście, takie jak kraj, w którym leży to miasto, przybliżona populacja oraz pewien fakt z historii tego miasta. Kluczami słownika zawierającego informacje o mieście mogą więc być `'country'`, `'population'` i `'fact'`. Wyświetl nazwę każdego miasta oraz wszystkie zebrane o nim informacje.

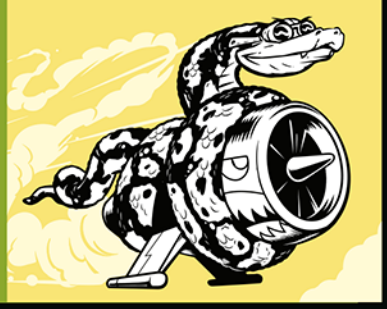
6.12. Rozszerzenia. Mamy już do czynienia z przykładami skomplikowanymi na tyle, że można je rozbudowywać na wiele różnych sposobów. Wykorzystaj jeden z przykładów przedstawionych w tym rozdziale i rozbuduj go, dodając nowe klucze i wartości, zmieniając kontekst programu lub poprawiając formatowanie danych wyjściowych.

Podsumowanie

W tym rozdziale dowiedziałeś się, jak zdefiniować słownik i pracować z umieszczonymi w nim informacjami. Zobaczyłeś, jak uzyskać dostęp do poszczególnych elementów słownika i je modyfikować, a także jak przeprowadzać iterację przez wszystkie informacje znajdujące się w słowniku. Nauczyłeś się przeprowadzać iteracje zarówno przez pary klucz-wartość słownika, jak i przez same jego klucze i wartości. Ponadto dowiedziałeś się, jak zagnieżdżać wiele słowników na jednej liście, wiele list w jednym słowniku oraz słowniki wewnątrz innego słownika.

W następnym rozdziale poznasz pętlę `while` i zobaczysz, jak akceptować dane wejściowe pochodzące od użytkowników programu. To będzie naprawdę ekscytujący rozdział, ponieważ nauczysz się zapewniać interaktywność tworzonym programom, które wreszcie będą mogły reagować na dane wejściowe wprowadzane przez użytkowników.


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



ZACZNIJ PROGRAMOWAĆ W PYTHONIE! SZYBKO, JUŻ!

Popularność Pythona stale rośnie: jest wszechstronny i zoptymalizowany pod kątem efektywności pracy, czytelności kodu i jakości oprogramowania, do tego darmowy, łatwo przenośny i można się go szybko nauczyć. Nadaje się do tworzenia gier i aplikacji sieciowych, do wdrażania indywidualnych rozwiązań biznesowych, sprawdza się nawet jako bezcenne narzędzie badaczy różnych dziedzin nauki. Jeśli tylko programista trochę się postara, może w Pythonie łatwo pisać przejrzysty, zwięzły kod, który jest prosty w utrzymaniu i nie sprawia problemów przy rozwijaniu oprogramowania. Python to idealny wybór dla każdego, kto nie chce tracić zbyt dużo czasu na naukę i liczy na to, że szybko zacznie pisać poprawny i działający kod.

To drugie, zaktualizowane i poprawione wydanie bestsellerowego podręcznika programowania w Pythonie pozwoli Ci błyskawicznie zacząć tworzyć kod, który działa! Zaczniesz od zrozumienia podstawowych koncepcji programistycznych, następnie nauczysz się zapewniać programom interaktywność i wykształcisz nawyk starannego testowania kodu przed wdrożeniem. Poszczególne zagadnienia będziesz natychmiast utrwalać dzięki licznym ćwiczeniom. Kolejnym etapem nauki będą praktyczne projekty: gra zręcznościowa, wizualizacja danych oraz aplikacja internetowa. Umiejętności, które zdobędziesz w ramach tego błyskawicznego kursu Pythona, pozwolą Ci tworzyć własne, rzeczywiste i wykorzystywane w praktyce aplikacje!

W tej książce znajdziesz dokładne instrukcje, jak:

- przygotować środowisko pracy i napisać swój pierwszy program
- wykorzystywać biblioteki i narzędzia Pythona, w tym pygame, matplotlib, plotly i Django
- generować interaktywne wizualizacje danych
- tworzyć proste aplikacje internetowe i wdrażać je na serwerach WWW
- testować i debugować kod oraz z powodzeniem rozwiązywać pojawiające się problemy

Eric Matthes — znawca systemu Linux. Od 15 lat pracuje z różnymi dystrybucjami oraz programuje w różnych językach. W branży IT działa od ponad 30 lat. Jest twórcą popularnego serwisu LinuxCommand.org.

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

KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-6360-1
9 788328 363601
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 99,00 zł
no starch press