

Wydanie III

PYTHON

INSTRUKCJE
DLA PROGRAMISTY

ERIC MATTHES



no starch
press

Helion

Tytuł oryginału: Python Crash Course: A Hands-On, Project-Based Introduction to Programming, 3rd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-0430-9

Copyright © 2023 by Eric Matthes. Title of English-language original: Python Crash Course: A Hands-On, Project-Based Introduction to Programming, 3rd Edition, ISBN 9781718502703, published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103.

The Polish-language 3rd edition Copyright © 2023 by Helion S.A. under license by No Starch Press Inc. All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/pytip3>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

WPROWADZENIE DO TRZECIEGO WYDANIA KSIĄŻKI	21
PODZIĘKOWANIA	24
WPROWADZENIE	26
CZĘŚĆ I. PODSTAWY	31
1	
ROZPOCZĘCIE PRACY	33
Przygotowanie środowiska programistycznego	33
Wersje Pythona	33
Wykonanie fragmentu kodu w Pythonie	34
Edytor tekstu VS Code	34
Python w różnych systemach operacyjnych	35
Python w systemie Windows	35
Python w systemie macOS	37
Python w systemach z rodziny Linux	39
Uruchomienie programu typu „Witaj, świecie!”	40
Instalacja w VS Code rozszerzenia przeznaczonego do obsługi Pythona	40
Uruchomienie programu typu „Witaj, świecie!”	41
Rozwiązywanie problemów podczas instalacji	42
Uruchamianie programów Pythona z poziomu powłoki	43
W systemie Windows	43
W systemach macOS i Linux	44
Podsumowanie	45
2	
ZMIENNE I PROSTE TYPY DANYCH	46
Co tak naprawdę dzieje się po uruchomieniu hello_world.py?	46
Zmienne	47
Nadawanie nazw zmiennym i używanie zmiennych	48
Unikanie błędów związanych z nazwami podczas używania zmiennych	49
Zmienna to etykieta	50

Ciągi tekstowe	51
Zmiana wielkości liter ciągu tekstowego za pomocą metod	51
Używanie zmiennych w ciągach tekstowych	53
Dodawanie białych znaków do ciągów tekstowych za pomocą tabulatora i znaku nowego wiersza	54
Usunięcie białych znaków	55
Usunięcie prefiksu	56
Unikanie błędów składni w ciągach tekstowych	57
Liczby	59
Liczby całkowite	59
Liczby zmiennoprzecinkowe	60
Liczby całkowite i zmiennoprzecinkowe	61
Znaki podkreślenia w liczbach	61
Wiele przypisań	62
Stałe	62
Komentarze	63
Jak można utworzyć komentarz?	63
Jakiego rodzaju komentarze należy tworzyć?	63
Zen Pythona	64
Podsumowanie	66

3

WPROWADZENIE DO LIST	67
Czym jest lista?	67
Uzyskanie dostępu do elementów listy	68
Numeracja indeksu zaczyna się od 0, a nie od 1	68
Użycie poszczególnych wartości listy	69
Zmienianie, dodawanie i usuwanie elementów	70
Modyfikowanie elementów na liście	70
Dodawanie elementów do listy	71
Usuwanie elementu z listy	73
Organizacja listy	77
Trwałe sortowanie listy za pomocą metody sort()	78
Tymczasowe sortowanie listy za pomocą funkcji sorted()	78
Wyświetlanie listy w odwrotnej kolejności alfabetycznej	80
Określenie wielkości listy	80
Unikanie błędów indeksu podczas pracy z listą	81
Podsumowanie	83

4

PRACA Z LISTĄ	84
Iteracja przez całą listę	84
Dokładniejsza analiza pętli	85
Wykonanie większej liczby zadań w pętli for	86
Wykonywanie operacji po pętli for	88

Unikanie błędów związanych z wcięciami	89
Brak wcięcia	89
Brak wcięcia dodatkowych wierszy	90
Niepotrzebne wcięcia	90
Niepotrzebne wcięcia po pętli	91
Brak dwukropka	92
Tworzenie list liczbowych	93
Użycie funkcji range()	93
Użycie funkcji range() do utworzenia listy liczb	94
Proste dane statystyczne dotyczące listy liczb	96
Lista składana	96
Praca z fragmentami listy	98
Wycinek listy	98
Iteracja przez wycinek	100
Kopiowanie listy	101
Krotka	104
Definiowanie krotki	104
Iteracja przez wszystkie wartości krotki	105
Nadpisanie krotki	105
Styl tworzonego kodu	106
Konwencje stylu	107
Wcięcia	107
Długość wiersza	108
Puste wiersze	108
Inne specyfikacje stylu	108
Podsumowanie	109
5	
KONSTRUKCJA IF	110
Prosty przykład	110
Test warunkowy	111
Sprawdzenie równości	111
Ignorowanie wielkości liter podczas sprawdzania równości	112
Sprawdzenie nierówności	113
Porównania liczbowe	114
Sprawdzanie wielu warunków	115
Sprawdzanie, czy wartość znajduje się na liście	116
Sprawdzanie, czy wartość nie znajduje się na liście	116
Wyrażenie boolowskie	117
Polecenie if	118
Proste polecenia if	118
Polecenia if-else	119
Konstrukcja if-elif-else	120
Użycie wielu bloków elif	122
Pominięcie bloku else	123
Sprawdzanie wielu warunków	123

Używanie poleceń if z listami	126
Sprawdzanie pod kątem wartości specjalnych	126
Sprawdzanie, czy lista nie jest pusta	128
Użycie wielu list	129
Nadawanie stylu poleceniom if	131
Podsumowanie	132

6

SŁOWNIKI133

Prosty słownik	133
Praca ze słownikami	134
Uzyskiwanie dostępu do wartości słownika	135
Dodanie nowej pary klucz-wartość	136
Rozpoczęcie pracy od pustego słownika	137
Modyfikowanie wartości słownika	137
Usuwanie pary klucz-wartość	139
Słownik podobnych obiektów	139
Używanie metody get() w celu uzyskania dostępu do wartości	141
Iteracja przez słownik	143
Iteracja przez wszystkie pary klucz-wartość	143
Iteracja przez wszystkie klucze słownika	145
Iteracja przez uporządkowane klucze słownika	147
Iteracja przez wszystkie wartości słownika	148
Zagnieżdżanie	150
Lista słowników	150
Lista w słowniku	153
Słownik w słowniku	156
Podsumowanie	158

7

DANE WEJŚCIOWE UŻYTKOWNIKA I PĘTLA WHILE159

Jak działa funkcja input()?	160
Przygotowanie jasnych i zrozumiałych komunikatów	160
Użycie funkcji int() do akceptowania liczbowych danych wejściowych	162
Operator modulo	163
Wprowadzenie do pętli while	164
Pętla while w działaniu	165
Umożliwienie użytkownikowi podjęcia decyzji o zakończeniu działania programu	165
Użycie flagi	167
Użycie polecenia break do opuszczenia pętli	169
Użycie polecenia continue w pętli	170
Unikanie pętli działającej w nieskończoność	170

Użycie pętli while wraz z listami i słownikami	172
Przenoszenie elementów z jednej listy na drugą	172
Usuwanie z listy wszystkich egzemplarzy określonej wartości	174
Umieszczenie w słowniku danych wejściowych wprowadzonych przez użytkownika	174
Podsumowanie	176

8

FUNKCJE	177
Definiowanie funkcji	177
Przekazywanie informacji do funkcji	178
Argumenty i parametry	179
Przekazywanie argumentów	180
Argumenty pozycyjne	180
Argumenty w postaci słów kluczowych	182
Wartości domyślne	183
Odpowiedniki wywołań funkcji	184
Unikanie błędów związanych z argumentami	185
Wartość zwrotna	186
Zwrot prostej wartości	187
Definiowanie argumentu jako opcjonalnego	187
Zwrot słownika	189
Używanie funkcji wraz z pętlą while	190
Przekazywanie listy	193
Modyfikowanie listy w funkcji	193
Uniemożliwianie modyfikowania listy przez funkcję	196
Przekazywanie dowolnej liczby argumentów	197
Argumenty pozycyjne i przekazywanie dowolnej liczby argumentów	199
Używanie dowolnej liczby argumentów w postaci słów kluczowych	200
Przechowywanie funkcji w modułach	202
Import całego modułu	202
Import określonych funkcji	203
Użycie słowa kluczowego as w celu zdefiniowania aliasu funkcji	204
Użycie słowa kluczowego as w celu zdefiniowania aliasu modułu	205
Import wszystkich funkcji modułu	205
Nadawanie stylu funkcjom	206
Podsumowanie	207

9

KLASY	209
Utworzenie i użycie klasy	210
Utworzenie klasy Dog	210
Metoda __init__()	211
Utworzenie egzemplarza na podstawie klasy	212

Praca z klasami i egzemplarzami	215
Klasa Car	215
Przypisanie atrybutowi wartości domyślnej	216
Modyfikacja wartości atrybutu	217
Dziedziczenie	221
Metoda <code>__init__()</code> w klasie potomnej	221
Definiowanie atrybutów i metod dla klasy potomnej	223
Nadpisywanie metod klasy nadrzędnej	225
Egzemplarz jako atrybut	225
Modelowanie rzeczywistych obiektów	228
Import klas	229
Import pojedynczej klasy	229
Przechowywanie wielu klas w module	231
Import wielu klas z modułu	232
Import całego modułu	233
Import wszystkich klas z modułu	234
Import modułu w module	234
Używanie aliasów	235
Określenie swojego sposobu pracy	236
Biblioteka standardowa Pythona	237
Nadawanie stylu klasom	238
Podsumowanie	239

10

PLIKI I WYJĄTKI	240
Odczytywanie danych z pliku	241
Wczytywanie całego pliku	241
Względna i bezwzględna ścieżka dostępu do pliku	243
Odczytywanie wiersz po wierszu	244
Praca z zawartością pliku	245
Ogromne pliki, czyli na przykład milion cyfr	246
Czy data Twoich urodzin znajduje się w liczbie pi?	247
Zapisywanie danych w pliku	249
Zapisywanie pojedynczego wiersza	249
Zapisywanie wielu wierszy	249
Wyjątki	251
Obsługiwanie wyjątku <code>ZeroDivisionError</code>	251
Używanie bloku <code>try-except</code>	252
Używanie wyjątków w celu uniknięcia awarii programu	252
Blok <code>else</code>	253
Obsługa wyjątku <code>FileNotFoundError</code>	255
Analiza tekstu	256
Praca z wieloma plikami	257
Ciche niepowodzenie	259
Które błędy należy zgłaszać?	260

Przechowywanie danych	261
Używanie json.dumps() i json.loads()	262
Zapisywanie i odczytywanie danych wygenerowanych przez użytkownika	263
Refaktoryzacja	266
Podsumowanie	269

11

TESTOWANIE KODU	270
Instalowanie pytest za pomocą pip	271
Uaktualnianie pip	271
Instalowanie pytest	272
Testowanie funkcji	272
Test jednostkowy i zestaw testów	274
Zaliczenie testu	274
Wykonywanie testu	275
Niezaliczenie testu	276
Reakcja na niezaliczony test	277
Dodanie nowego testu	279
Testowanie klasy	280
Różne rodzaje metod asercji	280
Klasa do przetestowania	281
Testowanie klasy AnonymousSurvey	283
Używanie danych testowych	285
Podsumowanie	287

CZĘŚĆ II. PROJEKTY289

12

STATEK, KTÓRY STRZELA POCISKAMI	291
Planowanie projektu	292
Instalacja Pygame	292
Rozpoczęcie pracy nad projektem gry	293
Utworzenie okna Pygame i reagowanie na działania użytkownika	293
Określenie liczby klatek na sekundę	295
Zdefiniowanie koloru tła	296
Utworzenie klasy ustawień	297
Dodanie obrazu statku kosmicznego	298
Utworzenie klasy statku kosmicznego	299
Wyświetlenie statku kosmicznego na ekranie	301
Refaktoryzacja, czyli metody _check_events() i _update_screen()	302
Metoda _check_events()	303
Metoda _update_screen()	303

Kierowanie statkiem kosmicznym	304
Reakcja na naciśnięcie klawisza	305
Umożliwienie nieustannego ruchu	305
Poruszanie statkiem w obu kierunkach	307
Dostosowanie szybkości statku	309
Ograniczenie zasięgu poruszania się statku	310
Refaktoryzacja metody <code>_check_events()</code>	311
Naciśnięcie klawisza Q w celu zakończenia gry	312
Uruchamianie gry w trybie pełnoekranowym	312
Krótkie powtórzenie	313
<code>alien_invasion.py</code>	313
<code>settings.py</code>	314
<code>ship.py</code>	314
Wystrzeliwanie pocisków	315
Dodawanie ustawień dotyczących pocisków	315
Utworzenie klasy <code>Bullet</code>	315
Przechowywanie pocisków w grupie	317
Wystrzeliwanie pocisków	318
Usuwanie niewidocznych pocisków	320
Ograniczenie liczby pocisków	321
Utworzenie metody <code>_update_bullets()</code>	321
Podsumowanie	323
13	
OBCY!	324
Przegląd projektu	325
Utworzenie pierwszego obcego	325
Utworzenie klasy <code>Alien</code>	326
Utworzenie egzemplarza obcego	327
Utworzenie floty obcych	328
Utworzenie rzędów obcych	329
Refaktoryzacja metody <code>_create_fleet()</code>	330
Dodawanie rzędów	332
Poruszanie flotą obcych	334
Przesunięcie obcych w prawo	334
Zdefiniowanie ustawień dla kierunku poruszania się floty	336
Sprawdzenie, czy obcy dotarł do krawędzi ekranu	336
Przesunięcie floty w dół i zmiana kierunku	337
Zestrzeliwanie obcych	339
Wykrywanie kolizji z pociskiem	339
Utworzenie większych pocisków w celach testowych	340
Ponowne utworzenie floty	340
Zwiększenie szybkości pocisku	342
Refaktoryzacja metody <code>_update_bullets()</code>	342

Zakończenie gry	343
Wykrywanie kolizji między obcym i statkiem	343
Reakcja na kolizję między obcym i statkiem	344
Obcy, który dociera do dolnej krawędzi ekranu	347
Koniec gry!	348
Ustalenie, które komponenty gry powinny być uruchomione	349
Podsumowanie	350

14

PUNKTACJA	351
Dodanie przycisku Gra	351
Utworzenie klasy Button	352
Wyświetlenie przycisku na ekranie	354
Uruchomienie gry	355
Zerowanie gry	356
Dezaktywacja przycisku Gra	357
Ukrycie kursora myszy	357
Zmiana poziomu trudności	358
Zmiana ustawień dotyczących szybkości	359
Wyzerowanie szybkości	360
Punktacja	361
Wyświetlanie punktacji	362
Utworzenie tablicy wyników	363
Uaktualnienie punktacji po zestrzeleniu obcego	364
Zerowanie wyniku	366
Zagwarantowanie uwzględnienia wszystkich trafień	366
Zwiększenie liczby zdobywanych punktów	367
Zaokrąglanie punktacji	368
Najlepsze wyniki	369
Wyświetlenie aktualnego poziomu gry	371
Wyświetlenie liczby statków	375
Podsumowanie	378

15

GENEROWANIE DANYCH	379
Instalacja matplotlib	380
Wygenerowanie prostego wykresu liniowego	380
Zmianianie etykiety i grubości wykresu	381
Poprawianie wykresu	383
Używanie wbudowanych stylów	384
Używanie funkcji scatter() do wyświetlania poszczególnych punktów i nadawania im stylu	386
Wyświetlanie serii punktów za pomocą funkcji scatter()	387
Automatyczne obliczanie danych	388
Dostosowanie znaczników osi do własnych potrzeb	389

Definiowanie własnych kolorów	390
Użycie mapy kolorów	390
Automatyczny zapis wykresu	392
Błądzenie losowe	392
Utworzenie klasy RandomWalk	392
Wybór kierunku	393
Wyświetlenie wykresu błądzenia losowego	394
Wygenerowanie wielu błędzeń losowych	395
Nadawanie stylu danym wygenerowanym przez błądzenie losowe	396
Symulacja rzutu kością do gry za pomocą plotly	402
Instalacja plotly	402
Utworzenie klasy Die	403
Rzut kością do gry	403
Analiza wyników	404
Utworzenie histogramu	405
Dostosowanie wykresu do własnych potrzeb	406
Rzut dwiema kośćmi	407
Dalsze dostosowanie wykresu do własnych potrzeb	409
Rzut kośćmi o różnej liczbie ścianek	410
Zapisywanie wykresu	411
Podsumowanie	412
16	
POBIERANIE DANYCH	413
Format CSV	414
Przetwarzanie nagłówków pliku CSV	414
Wyświetlanie nagłówków i ich położenia	415
Wyodrębnienie i odczytanie danych	416
Wyświetlenie danych na wykresie temperatury	417
Moduł datetime	418
Wyświetlanie daty	419
Wyświetlenie dłuższego przedziału czasu	420
Wyświetlenie drugiej serii danych	421
Nakładanie cienia na wykresie	423
Sprawdzenie pod kątem błędów	424
Samodzielne pobieranie danych	428
Mapowanie globalnych zbiorów danych – format GeoJSON	429
Pobranie danych dotyczących trzęsień ziemi	430
Analiza danych GeoJSON	430
Utworzenie listy trzęsień ziemi	433
Wyodrębnienie siły trzęsienia ziemi	433
Wyodrębnienie danych o miejscu wystąpienia trzęsienia ziemi	434
Budowanie mapy świata	435
Przedstawienie siły trzęsienia ziemi	436

Dostosowanie koloru punktu	437
Inne skale kolorów	439
Dodanie tekstu wyświetlanego po wskazaniu punktu na mapie	439
Podsumowanie	441

17

PRACA Z API 442

Użycie API	442
Git i GitHub	443
Żądanie danych za pomocą wywołania API	443
Instalacja requests	444
Przetworzenie odpowiedzi API	445
Praca ze słownikiem odpowiedzi	446
Podsumowanie repozytoriów najczęściej oznaczanych gwiazdką	449
Monitorowanie ograniczeń liczby wywołań API	450
Wizualizacja repozytoriów za pomocą pakietu plotly	451
Nadawanie stylu wykresowi	452
Dodanie własnych podpowiedzi	454
Dodawanie łączy do wykresu	455
Dostosowanie kolorów znaczników do własnych potrzeb	456
Więcej o plotly i API GitHub	457
Hacker News API	457
Podsumowanie	462

18

ROZPOCZĘCIE PRACY Z DJANGO 463

Przygotowanie projektu	464
Opracowanie specyfikacji	464
Utworzenie środowiska wirtualnego	464
Aktywacja środowiska wirtualnego	465
Instalacja frameworka Django	465
Utworzenie projektu w Django	466
Utworzenie bazy danych	467
Przegląd projektu	468
Uruchomienie aplikacji	469
Definiowanie modeli	470
Aktywacja modeli	471
Witryna administracyjna Django	473
Zdefiniowanie modelu Entry	476
Migracja modelu Entry	477
Rejestracja modelu Entry w witrynie administracyjnej	478
Powłoka Django	479
Tworzenie stron internetowych — strona główna aplikacji	481
Mapowanie adresu URL	482
Utworzenie widoku	484
Utworzenie szablonu	484

Utworzenie dodatkowych stron	486
Dziedziczenie szablonu	486
Strona tematów	489
Strony poszczególnych tematów	492
Podsumowanie	496

19

KONTA UŻYTKOWNIKÓW	497
Umożliwienie użytkownikom wprowadzania danych	497
Dodawanie nowego tematu	498
Dodawanie nowych wpisów	503
Edycja wpisu	507
Konfiguracja kont użytkowników	511
Aplikacja accounts	511
Strona logowania	512
Wylogowanie	515
Strona rejestracji użytkownika	516
Umożliwienie użytkownikom bycia właścicielami swoich danych	520
Ograniczenie dostępu za pomocą dekoratora @login_required	520
Powiązanie danych z określonymi użytkownikami	522
Przyznanie dostępu jedynie odpowiednim użytkownikom	525
Ochrona tematów użytkownika	526
Ochrona strony edit_entry	527
Powiązanie nowego tematu z bieżącym użytkownikiem	528
Podsumowanie	529

20

NADANIE STYLU I WDROŻENIE APLIKACJI	531
Nadanie stylu aplikacji Learning Log	532
Aplikacja django-bootstrap5	532
Użycie Bootstrapa do nadania stylu aplikacji Learning Log	533
Modyfikacja pliku base.html	533
Użycie elementu Jumbotron do nadania stylu stronie głównej	540
Nadanie stylu stronie logowania	541
Nadanie stylu stronie tematów	542
Nadanie stylów wpisom na stronie tematu	543
Wdrożenie aplikacji Learning Log	545
Utworzenie konta w Platform.sh	546
Instalacja Platform.sh CLI	546
Instalacja platformshconfig	546
Utworzenie pliku requirements.txt	547
Dodatkowe wymagania dotyczące wdrożenia	547
Dodawanie plików konfiguracyjnych	548
Modyfikacja pliku settings.py dla Platform.sh	552
Użycie Gita do monitorowania plików projektu	553

Utworzenie projektu w Platform.sh	555
Przekazanie projektu do Platform.sh	557
Wyświetlenie wdrożonego projektu	558
Dopracowanie wdrożenia projektu w Platform.sh	558
Utworzenie własnych stron błędu	561
Nieustanna rozbudowa	563
Usunięcie projektu z Platform.sh	564
Podsumowanie	565

A

INSTALACJA PYTHONA I ROZWIĄZYWANIE PROBLEMÓW567

Python w Windows	567
Użycie polecenia py zamiast python	568
Ponowna instalacja Pythona	568
Python w systemie macOS	568
Przypadkowa instalacja wersji dostarczanej przez Apple	569
Python 2 w starszych wydaniach systemu macOS	569
Python w systemie Linux	569
Używanie domyślnej instalacji Pythona	569
Instalacja najnowszej wersji Pythona	570
Sprawdzenie aktualnie używanej wersji Pythona	570
Słowa kluczowe Pythona i wbudowane funkcje	571
Słowa kluczowe Pythona	571
Wbudowane funkcje Pythona	571

B

EDYTORY TEKSTU I ŚRODOWISKA IDE573

Efektywna praca z VS Code	574
Konfigurowanie VS Code	575
Wybrane skróty klawiszowe VS Code	577
Inne edytory tekstu i środowiska IDE	579
IDLE	579
Geany	579
Sublime Text	579
Emacs i vim	579
PyCharm	580
Notatniki Jupyter Notebooks	580

C

UZYSKIWANIE POMOCY581

Pierwsze kroki	581
Spróbuj jeszcze raz	582
Chwila odpoczynku	582
Korzystaj z zasobów tej książki	583

Wyszukiwanie informacji w internecie	583
Stack Overflow	583
Oficjalna dokumentacja Pythona	584
Oficjalna dokumentacja biblioteki	584
r/learnpython	584
Posty na blogach	585
Discord	585
Slack	585
D	
UŻYWANIE GITA DO KONTROLI WERSJI	586
Instalacja Gita	587
Konfiguracja Gita	587
Tworzenie projektu	588
Ignorowanie plików	588
Inicjalizacja repozytorium	589
Sprawdzanie stanu	589
Dodawanie plików do repozytorium	590
Zatwierdzanie plików	590
Sprawdzanie dziennika projektu	591
Drugie zatwierdzenie	591
Przywracanie stanu projektu	593
Przywracanie projektu do poprzedniego stanu	594
Usuwanie repozytorium	596
E	
ROZWIĄZYWANIE PROBLEMÓW Z WDROŻENIAMI	598
Jak wygląda proces wdrażania?	598
Podstawy rozwiązywania problemów	599
Kierowanie się odpowiedziami wyświetlanymi na ekranie	600
Odczytywanie danych wyjściowych dzienników zdarzeń	601
Rozwiązywanie problemów dotyczących konkretnego systemu operacyjnego	603
Wdrażanie z poziomu systemu Windows	604
Wdrażanie z poziomu systemu macOS	605
Wdrażanie z poziomu systemu Linux	606
Inne podejścia w zakresie wdrożenia	607

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 otrzymanych 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']  
print(f"Zdobyłeś {new_points} punktów!")
```

Odwołując się do zdefiniowanego słownika, kod pobiera wartość przypisaną do klucza 'points'. Następnie ta wartość zostaje umieszczona w zmiennej `new_points`. Kod w ostatnim wierszu 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
alien_0['y_position'] = 25
print(alien_0)
```

Rozpoczynamy od zdefiniowania takiego samego słownika jak wcześniej. Następnie wyświetlamy jego zawartość, aby w ten sposób mieć punkt odniesienia. Później dodajemy do słownika nową parę klucz-wartość: klucz `'x_position'`, wartość 0. Do słownika dodajemy także 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.

Słownik zachowuje kolejność, w której były dodawane pary klucz-wartość. Podczas wyświetlania słownika lub iteracji przez jego elementy zostaną one wyświetlone w kolejności ich dodawania 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 przesuwany 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': 'rust',
    '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': 'rust',
    '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 pominiiesz 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, wcięтым 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:

Plik `user.py`:

```
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.

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

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

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 `for` mamy nazwę słownika oraz metodę `items()`, której wartością zwrótną jest lista par klucz-wartość. Następnie pętla `for` przechowuje poszczególne pary w dwóch przedstawionych zmiennych. W omawianym fragmencie kodu zmienne wykorzystujemy do wyświetlenia klucza oraz przypisanej mu wartości. 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': 'rust',
    'paweł': 'python',
}
```

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

W kodzie 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 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 Rust.
```

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': 'rust',
    'paweł': 'python',
}

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

W pętli `for` 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 przyjaciółom 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 = {
    --cięcie--
}

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

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

Rozpoczynamy od utworzenia listy przyjaciół, którym chcemy wyświetlić komunikat. Wewnątrz pętli wyświetlamy imiona wszystkich osób. Następnie w wierszu ❶ 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 ❷). Następnie wyświetlamy specjalne powitanie odwołujące się do ulubionego języka programowania.

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

```
Witaj, Edward.
Witaj, Paweł.
    Witaj, Paweł! Widzę, że Twoim ulubionym językiem programowania jest Python!
Witaj, Sara.
```

Witaj, Sara! Widzę, że Twoim ulubionym językiem programowania jest C!
Witaj, 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 = {
    --cięcie--
}

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

Metoda `keys()` nie służy jedynie do przeprowadzania iteracji. W rzeczywistości zwraca listę wszystkich kluczy, a polecenie `if` 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

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 kolejności, 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': 'rust',
    '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 zwrótu 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': 'rust',
    '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
Rust
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 = {
    --cięcie--
}
```

```
print("W ankiecie zostały wymienione następujące języki programowania:")
for language in set(favorite_languages.values()):
    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 omawianym kodzie wykorzystujemy wywołanie `set()` do pobrania unikatowych nazw języków otrzymanych jako wynik działania metody `favorite_languages.values()`.

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

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

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', 'rust', 'python', 'c'}
>>> languages
{'rust', '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 następujące 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)}")
```

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 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 postępowaniem 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 = {
    'crust': 'grubym',
    'toppings': ['pieczarki', 'podwójny ser'],
}

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

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

Rozpoczynamy od utworzenia 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 ❶ 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 ❷). 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 = {
    'janek': ['python', 'rust'],
    'sara': ['c'],
    'edward': ['rust', 'go'],
    'paweł': ['python', 'haskell'],
}

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

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 ❶) 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 ❷) 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
Rust
```

Ulubione języki programowania użytkownika Sara to:

```
C
```

Ulubione języki programowania użytkownika Edward to:

```
Rust
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 ③ 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 ④):

```
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 



POZNAJ PYTHONA – Z TYM ŚWIATOWYM BESTSELLEREM ZROBISZ TO NAJSZYBCIEJ!

Pythona można się szybko nauczyć i praktycznie od razu zacząć tworzyć działające gry, aplikacje internetowe czy też programy wspierające badaczy różnych dziedzin nauki. Język ten został pomyślany tak, aby ułatwiać pisanie przejrzystego, zwięzłego kodu w sposób zgodny ze sztuką programowania. To idealny wybór dla każdego, komu zależy, by nie tracić zbyt dużo czasu na naukę, tylko od razu pisać kod działający zgodnie z oczekiwaniami.

To trzecie, zaktualizowane i uzupełnione wydanie bestsellerowego podręcznika programowania w Pythonie. Naukę rozpoczniesz od podstawowych koncepcji programowania. Poznasz takie pojęcia jak zmienne, listy, klasy i pętle, a następnie utrwalisz je dzięki praktycznym ćwiczeniom. Dowiesz się, jak zapewnić interaktywność programom, i nauczysz się poprawnego testowania kodu przed dodaniem go do projektu. W kolejnych rozdziałach przystąpisz do praktycznej realizacji trzech projektów: gry zręcznościowej inspirowanej klasyczną Space Invaders, wizualizacji danych za pomocą dostępnych dla Pythona niezwykle użytecznych bibliotek i prostej aplikacji internetowej, gotowej do wdrożenia na serwerze WWW i opublikowania w internecie.

Skorzystaj z dokładnych instrukcji, jak:

- przygotować się do napisania pierwszego programu w Pythonie
- używać bibliotek i narzędzi Pythona
- pracować z różnymi zbiorami danych
- tworzyć interaktywne aplikacje internetowe i bezpiecznie je wdrażać
- testować i debugować kod, a także rozwiązywać różne problemy

Eric Matthes ma 25-letnie doświadczenie w dydaktyce. Był nauczycielem i prowadził kursy z zakresu Pythona. Obecnie jest programistą i angażuje się w rozwój wielu różnorodnych projektów open source. Wolny czas spędza z rodziną lub poświęca swojej pasji – wspinaczce górskiej.

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 978-83-289-0430-9



Cena: 119,00 zł

