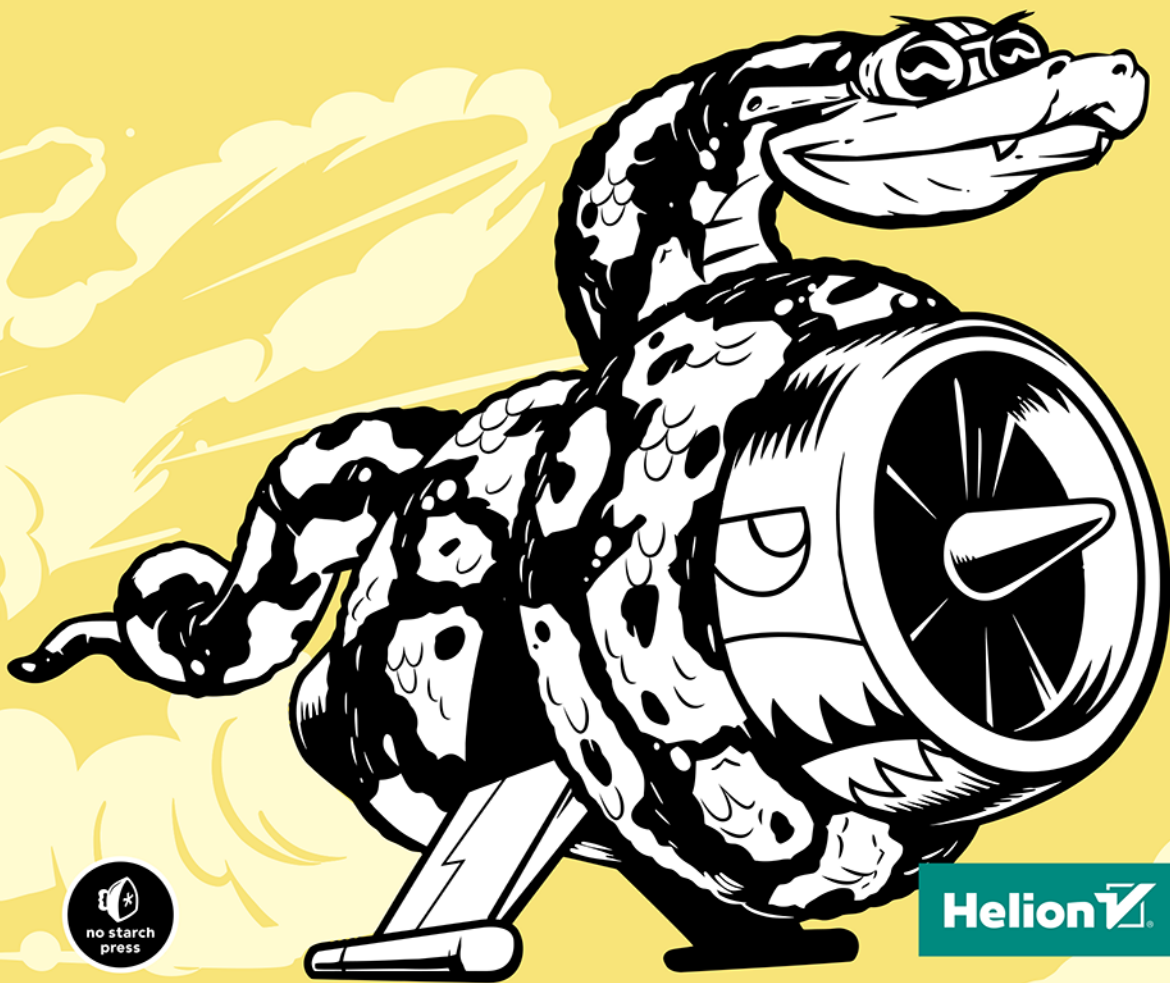


PYTHON

INSTRUKCJE DLA PROGRAMISTY

ERIC MATTHES



Helion

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

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-2595-1

Copyright © 2016 by Eric Matthes. Title of English-language original: Python Crash Course, ISBN 978-1-59327-603-4, published by No Starch Press.

Polish-language edition copyright © 2016 by Helion S.A.
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 Wydawnictwo HELION 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 Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
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/pythip.zip>

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

Printed in Poland.

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

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

Spis treści

O AUTORZE	21
O KOREKTORZE MERYTORYCZNYM	21
PODZIĘKOWANIA	23
WPROWADZENIE	25
Do kogo jest skierowana ta książka?	26
Czego nauczysz się z tej książki?	26
Dlaczego Python?	27
CZĘŚĆ I. PODSTAWY	29
I	
ROZPOCZĘCIE PRACY	31
Przygotowanie środowiska programistycznego	31
Python 2 i Python 3	31
Wykonanie fragmentu kodu w Pythonie	32
Witaj, świecie!	32
Python w różnych systemach operacyjnych	33
Python w systemach z rodziny Linux	33
Python w systemie OS X	37
Python w systemie Windows	40
Rozwiązywanie problemów podczas instalacji	44
Uruchamianie programów Pythona z poziomu powłoki	45
W systemach Linux i OS X	45
W systemie Windows	46
Podsumowanie	47

2

ZMIENNE I PROSTE TYPY DANYCH	49
Co tak naprawdę dzieje się po uruchomieniu <code>hello_world.py</code>	49
Zmienne	50
Nadawanie nazw zmiennym i używanie zmiennych	51
Unikanie błędów związanych z nazwami podczas używania zmiennych	52
Ciągi tekstowe	54
Zmiana wielkości liter ciągu tekstowego za pomocą metod	54
Łączenie ciągów tekstowych	55
Dodawanie białych znaków do ciągów tekstowych za pomocą tabulatora i znaku nowego wiersza	57
Usunięcie białych znaków	57
Unikanie błędów składni w ciągach tekstowych	59
Wyświetlanie danych w Pythonie 2.x	60
Liczby	61
Liczby całkowite	61
Liczby zmiennoprzecinkowe	62
Unikanie błędów typu podczas pracy z funkcją <code>str()</code>	63
Liczby całkowite w Pythonie 2	64
Komentarze	65
Jak można utworzyć komentarz?	65
Jakiego rodzaju komentarze należy tworzyć?	66
Zen Pythona	66
Podsumowanie	68

3

WPROWADZENIE DO LIST	71
Czym jest lista?	71
Uzyskanie dostępu do elementów listy	72
Numeracja indeksu zaczyna się od 0, a nie od 1	73
Użycie poszczególnych wartości listy	73
Zmienianie, dodawanie i usuwanie elementów	74
Modyfikowanie elementów na liście	75
Dodawanie elementów do listy	75
Usuwanie elementu z listy	77
Organizacja listy	82
Trwałe sortowanie listy za pomocą metody <code>sort()</code>	82
Wyświetlanie listy w odwrotnej kolejności alfabetycznej	84
Określenie wielkości listy	84
Unikanie błędów indeksu podczas pracy z listą	86
Podsumowanie	87

4

PRACA Z LISTĄ	89
Iteracja przez całą listę	89
Dokładniejsza analiza pętli	90
Wykonanie większej liczby zadań w pętli for	91
Wykonywanie operacji po pętli for	93
Unikanie błędów związanych z wcięciami	94
Brak wcięcia	94
Brak wcięcia dodatkowych wierszy	95
Niepotrzebne wcięcie	95
Niepotrzebne wcięcie po pętli	96
Brak dwukropka	97
Tworzenie list liczbowych	97
Użycie funkcji range()	97
Użycie funkcji range() do utworzenia listy liczb	99
Proste dane statystyczne dotyczące listy liczb	101
Lista składana	101
Praca z fragmentami listy	103
Wycinek listy	103
Iteracja przez wycinek	104
Kopiowanie listy	105
Krotka	108
Definiowanie krotki	108
Iteracja przez wszystkie wartości krotki	109
Nadpisanie krotki	110
Styl tworzonego kodu	111
Konwencje stylu	111
Wcięcia	112
Długość wiersza	112
Puste wiersze	113
Inne specyfikacje stylu	113
Podsumowanie	114

5

KONSTRUKCJA IF	115
Prosty przykład	115
Test warunkowy	116
Sprawdzenie równości	116
Ignorowanie wielkości liter podczas sprawdzania równości	117
Sprawdzenie nierówności	118
Porównania liczbowe	119
Sprawdzanie wielu warunków	120
Sprawdzanie, czy wartość znajduje się na liście	121
Sprawdzanie, czy wartość nie znajduje się na liście	122
Wyrażenie boolowskie	122

Polecenie if	123
Proste polecenia if	123
Polecenia if-else	125
Łącuch if-elif-else	125
Użycie wielu bloków elif	127
Pominięcie bloku else	128
Sprawdzanie wielu warunków	128
Używanie poleceń if z listami	132
Sprawdzanie pod kątem wartości specjalnych	132
Sprawdzanie, czy lista nie jest pusta	133
Użycie wielu list	134
Nadawanie stylu poleceniom if	136
Podsumowanie	137

6

SŁOWNIKI 139

Prosty słownik	140
Praca ze słownikami	140
Uzyskiwanie dostępu do wartości słownika	141
Dodanie nowej pary klucz-wartość	142
Rozpoczęcie pracy od pustego słownika	143
Modyfikowanie wartości słownika	143
Usuwanie pary klucz-wartość	145
Słownik podobnych obiektów	145
Iteracja przez słownik	148
Iteracja przez wszystkie pary klucz-wartość	148
Iteracja przez wszystkie klucze słownika	150
Iteracja przez uporządkowane klucze słownika	152
Iteracja przez wszystkie wartości słownika	153
Zagnieżdżanie	154
Lista słowników	155
Lista w słowniku	158
Słownik w słowniku	161
Podsumowanie	163

7

DANE WEJŚCIOWE UŻYTKOWNIKA I PĘTLA WHILE 165

Jak działa funkcja input()?	166
Przygotowanie jasnych i zrozumiałych komunikatów	167
Użycie funkcji int() do akceptowania liczbowych danych wejściowych	168
Operator modulo	169
Akceptacja danych wejściowych w Pythonie 2.7	170
Wprowadzenie do pętli while	171
Pętla while w działaniu	171
Umożliwienie użytkownikowi podjęcia decyzji o zakończeniu działania programu	172

Użycie flagi	174
Użycie polecenia break do opuszczenia pętli	175
Użycie polecenia continue w pętli	176
Unikanie pętli działającej w nieskończoność	177
Użycie pętli while wraz z listami i słownikami	179
Przenoszenie elementów z jednej listy na drugą	179
Usuwanie z listy wszystkich egzemplarzy określonej wartości	180
Umieszczenie w słowniku danych wejściowych wprowadzonych przez użytkownika ...	181
Podsumowanie	183

8

FUNKCJE 185

Definiowanie funkcji	185
Przekazywanie informacji do funkcji	186
Argumenty i parametry	187
Przekazywanie argumentów	188
Argumenty pozycyjne	188
Argumenty w postaci słów kluczowych	190
Wartości domyślne	191
Odpowiedniki wywołań funkcji	192
Unikanie błędów związanych z argumentami	193
Wartość zwrotna	195
Zwrot prostej wartości	195
Definiowanie argumentu jako opcjonalnego	196
Zwrot słownika	198
Używanie funkcji wraz z pętlą while	199
Przekazywanie listy	200
Modyfikowanie listy w funkcji	202
Uniemożliwianie modyfikowania listy przez funkcję	205
Przekazywanie dowolnej liczby argumentów	205
Argumenty pozycyjne i przekazywanie dowolnej liczby argumentów	207
Używanie dowolnej liczby argumentów w postaci słów kluczowych	208
Przechowywanie funkcji w modułach	209
Import całego modułu	210
Import określonych funkcji	212
Użycie słowa kluczowego as w celu zdefiniowania aliasu funkcji	212
Użycie słowa kluczowego as w celu zdefiniowania aliasu modułu	213
Import wszystkich funkcji modułu	213
Nadawanie stylu funkcjom	214
Podsumowanie	216

9

KLASY 217

Utworzenie i użycie klasy	218
Utworzenie klasy Dog	218
Utworzenie egzemplarza na podstawie klasy	220

Praca z klasami i egzemplarzami	223
Klasa Car	224
Przypisanie atrybutowi wartości domyślnej	225
Modyfikacja wartości atrybutu	226
Dziedziczenie	229
Metoda <code>__init__()</code> w klasie potomnej	230
Dziedziczenie w Pythonie 2.7	231
Definiowanie atrybutów i metod dla klasy potomnej	232
Nadpisywanie metod klasy nadrzędnej	233
Egzemplarz jako atrybut	234
Modelowanie rzeczywistych obiektów	236
Import klas	237
Import pojedynczej klasy	237
Przechowywanie wielu klas w module	240
Import wielu klas z modułu	241
Import całego modułu	242
Import wszystkich klas z modułu	242
Import modułu w module	243
Określenie swojego sposobu pracy	244
Biblioteka standardowa Pythona	245
Nadawanie stylu klasom	246
Podsumowanie	248

10

PLIKI I WYJĄTKI 249

Odczytywanie danych z pliku	250
Wczytywanie całego pliku	250
Ścieżka dostępu do pliku	252
Odczytywanie wiersz po wierszu	253
Utworzenie listy wierszy na podstawie zawartości pliku	255
Praca z zawartością pliku	255
Ogromne pliki, czyli na przykład milion cyfr	257
Czy data Twoich urodzin znajduje się w liczbie pi?	257
Zapisywanie danych w pliku	258
Zapisywanie danych do pustego pliku	259
Zapisywanie wielu wierszy	260
Dołączanie do pliku	261
Wyjątki	262
Obsługiwanie wyjątku <code>ZeroDivisionError</code>	262
Używanie bloku <code>try-except</code>	263
Używanie wyjątków w celu uniknięcia awarii programu	264
Blok <code>else</code>	265
Obsługa wyjątku <code>FileNotFoundError</code>	266
Analiza tekstu	267
Praca z wieloma plikami	269

Ciche niepowodzenie	270
Które błędy należy zgłaszać?	271
Przechowywanie danych	273
Używanie json.dump() i json.load()	273
Zapisywanie i odczytywanie danych wygenerowanych przez użytkownika	274
Refaktoryzacja	277
Podsumowanie	279

II

TESTOWANIE KODU	281
Testowanie funkcji	282
Test jednostkowy i zestaw testów	283
Zaliczenie testu	283
Niezaliczenie testu	285
Reakcja na niezaliczony test	286
Dodanie nowego testu	288
Testowanie klasy	290
Różne rodzaje metod asercji	290
Klasa do przetestowania	290
Testowanie klasy AnonymousSurvey	293
Metoda setUp()	295
Podsumowanie	297

CZĘŚĆ II. PROJEKTY

299

PROJEKT I. INWAZJA OBCYCH

I 2

STATEK, KTÓRY STRZELA POCISKAMI	303
Planowanie projektu	304
Instalacja Pygame	304
Instalacja pakietów Pythona za pomocą pip	305
Rozpoczęcie pracy nad projektem gry	310
Utworzenie okna Pygame i reagowanie na działania użytkownika	310
Zdefiniowanie koloru tła	311
Utworzenie klasy ustawień	312
Dodanie obrazu statku kosmicznego	314
Utworzenie klasy statku kosmicznego	314
Wyświetlenie statku kosmicznego na ekranie	316
Refaktoryzacja, czyli moduł game_functions	317
Funkcja check_events()	317
Funkcja update_screen()	319

Kierowanie statkiem kosmicznym	320
Reakcja na naciśnięcie klawisza	320
Umożliwienie nieustannego ruchu	321
Poruszanie statkiem w obu kierunkach	323
Dostosowanie szybkości statku	325
Ograniczenie zasięgu poruszania się statku	327
Refaktoryzacja funkcji check_events()	327
Krótkie powtórzenie	328
alien_invasion.py	328
settings.py	329
game_functions.py	329
ship.py	329
Wystrzeliwanie pocisków	330
Dodawanie ustawień dotyczących pocisków	330
Utworzenie klasy Bullet	330
Przechowywanie pocisków w grupie	332
Wystrzeliwanie pocisków	333
Usuwanie niewidocznych pocisków	334
Ograniczenie liczby pocisków	336
Utworzenie funkcji update_bullets()	337
Utworzenie funkcji fire_bullet()	337
Podsumowanie	338

I 3

OBCY! 339

Przegląd projektu	340
Utworzenie pierwszego obcego	341
Utworzenie klasy Alien	341
Utworzenie egzemplarza obcego	342
Wyświetlenie obcego na ekranie	343
Utworzenie floty obcych	343
Ustalenie maksymalnej liczby obcych wyświetlanych w jednym rzędzie	344
Utworzenie rzędów obcych	345
Utworzenie floty	346
Refaktoryzacja funkcji create_fleet()	348
Dodawanie rzędów	349
Poruszanie flotą obcych	351
Przesunięcie obcych w prawo	352
Zdefiniowanie ustawień dla kierunku poruszania się floty	353
Sprawdzenie, czy obcy dotarł do krawędzi ekranu	354
Przesunięcie floty w dół i zmiana kierunku	355
Zestrzeliwanie obcych	356
Wykrywanie kolizji z pociskiem	356
Utworzenie większych pocisków w celach testowych	357
Ponowne utworzenie floty	358

Zwiększenie szybkości pocisku	360
Refaktoryzacja funkcji update_bullets()	360
Zakończenie gry	361
Wykrywanie kolizji między obcym i statkiem	361
Reakcja na kolizję między obcym i statkiem	362
Obcy, który dociera do dolnej krawędzi ekranu	366
Koniec gry!	367
Ustalenie, które komponenty gry powinny być uruchomione	367
Podsumowanie	368

14

PUNKTACJA	369
Dodanie przycisku Gra	369
Utworzenie klasy Button	370
Wyświetlenie przycisku na ekranie	372
Uruchomienie gry	373
Zerowanie gry	374
Dezaktywacja przycisku Gra	376
Ukrycie kursora myszy	377
Zmiana poziomu trudności	377
Zmiana ustawień dotyczących szybkości	378
Wyzerowanie szybkości	380
Punktacja	381
Wyświetlanie punktacji	381
Utworzenie tablicy wyników	383
Uaktualnienie punktacji po zestrzeleniu obcego	384
Zagwarantowanie uwzględnienia wszystkich trafień	386
Zwiększenie liczby zdobywanych punktów	387
Zaokrąglenie punktacji	388
Najlepsze wyniki	389
Wyświetlenie aktualnego poziomu gry	391
Wyświetlenie liczby statków	396
Podsumowanie	399

PROJEKT 2. WIZUALIZACJA DANYCH

15

GENEROWANIE DANYCH	405
Instalacja matplotlib	406
Linux	406
OS X	407
Windows	407
Testowanie matplotlib	407
Galeria matplotlib	408

Wygenerowanie prostego wykresu liniowego	408
Zmianie etykiety i grubości wykresu	409
Poprawianie wykresu	410
Używanie funkcji scatter() do wyświetlania poszczególnych punktów i nadawania im stylu	411
Wyświetlanie serii punktów za pomocą funkcji scatter()	413
Automatyczne obliczanie danych	413
Usuwanie konturów z wyświetlanych punktów danych	415
Definiowanie własnych kolorów	415
Użycie mapy kolorów	415
Automatyczny zapis wykresu	416
Błądzenie losowe	417
Utworzenie klasy RandomWalk	417
Wybór kierunku	418
Wyświetlenie wykresu błędzenia losowego	419
Wygenerowanie wielu błędzeń losowych	420
Nadawanie stylu danym wygenerowanym przez błędzenie losowe	421
Kolorowanie punktów	421
Kolorowanie punktów początkowego i końcowego	422
Ukrywanie osi	423
Dodawanie punktów do wykresu	424
Zmianie wielkości wykresu, aby wypełnił ekran	424
Symulacja rzutu kością do gry za pomocą Pygal	426
Instalacja Pygal	427
Galeria Pygal	427
Utworzenie klasy Die	427
Rzut kością do gry	428
Analiza wyników	429
Utworzenie histogramu	430
Rzut dwiema kośćmi	431
Rzut kośćmi o różnej liczbie ścianek	433
Podsumowanie	434

16

POBIERANIE DANYCH 437

Format CSV	438
Przetwarzanie nagłówków pliku CSV	438
Wyświetlanie nagłówków i ich położenia	439
Wyodrębnienie i odczytanie danych	440
Wyświetlenie danych na wykresie temperatury	442
Moduł datetime	442
Wyświetlanie daty	444
Wyświetlenie dłuższego przedziału czasu	445
Wyświetlenie drugiej serii danych	446

Nakładanie cienia na wykresie	447
Sprawdzenie pod kątem błędów	448
Mapowanie globalnych zbiorów danych — format JSON	452
Pobranie danych dotyczących populacji świata	452
Wyodrębnienie interesujących nas danych	453
Konwersja ciągu tekstowego na wartość liczbową	454
Pobranie dwuznakowego kodu państwa	455
Budowanie mapy świata	458
Wyświetlenie danych liczbowych na mapie świata	459
Wyświetlenie pełnej mapy populacji	460
Grupowanie państw według populacji	463
Nadawanie stylu mapie świata w Pygal	464
Rozjaśnienie motywu graficznego	467
Podsumowanie	468

17

PRACA Z API 471

Użycie Web API	471
Git i GitHub	472
Żądanie danych za pomocą wywołania API	472
Instalacja requests	473
Przetworzenie odpowiedzi API	473
Praca ze słownikiem odpowiedzi	474
Podsumowanie repozytoriów najczęściej oznaczanych gwiazdką	477
Monitorowanie ograniczeń liczby wywołań API	478
Wizualizacja repozytoriów za pomocą pakietu Pygal	479
Dopracowanie wykresów generowanych przez Pygal	481
Dodanie własnych podpowiedzi	483
Wyświetlanie danych	484
Dodawanie łączy do wykresu	486
Hacker News API	487
Podsumowanie	490

PROJEKT 3. APLIKACJE SIECIOWE

18

ROZPOCZĘCIE PRACY Z DJANGO 495

Przygotowanie projektu	496
Opracowanie specyfikacji	496
Utworzenie środowiska wirtualnego	496
Instalacja virtualenv	497
Aktywacja środowiska wirtualnego	497
Instalacja frameworka Django	498
Utworzenie projektu w Django	498

Utworzenie bazy danych	499
Przegląd projektu	500
Uruchomienie aplikacji	501
Definiowanie modeli	502
Aktywacja modeli	503
Witryna administracyjna Django	505
Zdefiniowanie modelu Entry	508
Migracja modelu Entry	509
Rejestracja modelu Entry w witrynie administracyjnej	509
Powłoka Django	510
Tworzenie stron internetowych — strona główna aplikacji	512
Mapowanie adresu URL	513
Utworzenie widoku	515
Utworzenie szablonu	516
Utworzenie dodatkowych stron	517
Dziedziczenie szablonu	518
Strona tematów	520
Strony poszczególnych tematów	523
Podsumowanie	527

19

KONTA UŻYTKOWNIKÓW 529

Umożliwienie użytkownikom wprowadzania danych	530
Dodawanie nowego tematu	530
Dodawanie nowych wpisów	535
Edycja wpisu	539
Konfiguracja kont użytkowników	543
Aplikacja users	543
Strona logowania	544
Wylogowanie	547
Strona rejestracji użytkownika	548
Umożliwienie użytkownikom bycia właścicielami swoich danych	552
Ograniczenie dostępu za pomocą dekoratora @login_required	552
Powiązanie danych z określonymi użytkownikami	555
Przyznanie dostępu jedynie odpowiednim użytkownikom	558
Ochrona tematów użytkownika	559
Ochrona strony edit_entry	560
Powiązanie nowego tematu z bieżącym użytkownikiem	560
Podsumowanie	562

20

NADANIE STYLU I WDROŻENIE APLIKACJI 563

Nadanie stylu aplikacji Learning Log	564
Aplikacja django-bootstrap3	564
Użycie Bootstrapa do nadania stylu aplikacji Learning Log	565

Modyfikacja pliku base.html	566
Użycie elementu Jumbotron do nadania stylu stronie głównej	570
Nadanie stylu stronie logowania	570
Nadanie stylu stronie new_topic	571
Nadanie stylu stronie tematów	573
Nadanie stylów wpisom na stronie tematu	573
Wdrożenie aplikacji Learning Log	575
Utworzenie konta w Heroku	576
Instalacja Heroku Toolbelt	576
Instalacja wymaganych pakietów	576
Utworzenie listy pakietów w pliku requirements.txt	577
Określenie środowiska uruchomieniowego Pythona	578
Modyfikacja pliku settings.py dla Heroku	579
Utworzenie pliku Procfile do uruchomienia procesu	580
Modyfikacja pliku wsgi.py dla Heroku	580
Utworzenie katalogu dla plików statycznych	580
Użycie serwera gunicorn w środowisku lokalnym	581
Użycie Gita do monitorowania plików projektu	582
Przekazanie projektu do Heroku	584
Konfiguracja bazy danych w Heroku	585
Dopracowanie wdrożenia projektu w Heroku	586
Zabezpieczenie wdrożonego projektu	588
Zatwierdzenie zmian i przekazanie ich do serwera	589
Utworzenie własnych stron błędu	590
Nieustanna rozbudowa	593
Opcja SECRET_KEY	594
Usunięcie projektu z Heroku	594
Podsumowanie	595

POSŁOWIE 597

A

INSTALACJA PYTHONA 599

Python w systemie Linux	599
Ustalenie zainstalowanej wersji	599
Instalacja Pythona 3 w systemie Linux	600
Python w systemie OS X	600
Ustalenie zainstalowanej wersji	600
Użycie menedżera Homebrew do instalacji Pythona 3	601
Python w Windows	602
Instalacja Pythona 3 w Windows	602
Odszukanie interpretera Pythona	603
Dodanie Pythona do zmiennej Path	603

Słowa kluczowe Pythona i wbudowane funkcje	604
Słowa kluczowe Pythona	604
Wbudowane funkcje Pythona	604

B

EDYTORY TEKSTU 607

Geany	608
Instalacja Geany w systemie Linux	608
Instalacja Geany w systemie Windows	608
Uruchamianie programów Python w Geany	609
Dostosowanie ustawień Geany do własnych potrzeb	610
Sublime Text	610
Instalacja Sublime Text w systemie OS X	611
Instalacja Sublime Text w systemie Linux	611
Instalacja Sublime Text w systemie Windows	611
Uruchamianie programów Python w edytorze Sublime Text	611
Konfigurowanie edytora Sublime Text	612
Dostosowanie ustawień Sublime Text do własnych potrzeb	612
IDLE	613
Instalacja IDLE w systemie Linux	613
Instalacja IDLE w systemie OS X	613
Instalacja IDLE w systemie Windows	614
Dostosowanie ustawień IDLE do własnych potrzeb	614
Emacs i vim	614

C

UZYSKIWANIE POMOCY 615

Pierwsze kroki	615
Spróbuj jeszcze raz	616
Chwila odpoczynku	616
Korzystaj z zasobów tej książki	617
Wyszukiwanie informacji w internecie	617
Stack Overflow	617
Oficjalna dokumentacja Pythona	618
Oficjalna dokumentacja biblioteki	618
r/learnpython	618
Posty na blogach	618
Kanały IRC	618
Założenie konta na kanale IRC	619
Kanały, do których warto się przyłączyć	619
Kultura na kanale IRC	619

D**UŻYWANIE GITA DO KONTROLI WERSJI 621**

Instalacja Gita	622
Instalacja Gita w systemie Linux	622
Instalacja Gita w systemie OS X	622
Instalacja Gita w systemie Windows	622
Konfiguracja Gita	622
Tworzenie projektu	623
Ignorowanie plików	623
Inicjalizacja repozytorium	624
Sprawdzanie stanu	624
Dodawanie plików do repozytorium	625
Zatwierdzanie plików	625
Sprawdzanie dziennika projektu	626
Drugie zatwierdzenie	627
Przywracanie stanu projektu	628
Przywrócenie projektu do wcześniejszego stanu	629
Usunięcie repozytorium	631

SKOROWIDZ 633

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("Zdobyłeś " + str(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:

```
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. Zwróć uwagę na to, że kolejność par klucz-wartość nie musi odpowiadać kolejności ich dodawania do słownika. Ważne jest jedynie połączenie między kluczem i jego wartością.

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:

```
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:

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

alien_0['color'] = 'żółty'
print("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("Początkowa wartość x-position: " + str(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("Nowa wartość x-position: " + str(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:

```
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',
}

print("Ulubiony język programowania Sary to " + ❶
      favorite_languages['sara'].title() + ❷
      ".") ❸
```

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

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

Powyższa składnia została użyta w wywołaniu `print()` widocznym w wierszu ❷. Wygenerowane dane wyjściowe zawierają informacje o ulubionym języku programowania Sary, tak jak pokazałem poniżej:

Ulubiony język programowania Sary to C.

Ten przykład pokazuje również, jak można podzielić długie polecenie `print()` na wiele wierszy. Słowo `print` jest krótsze niż większość nazw słowników, więc sensowne jest umieszczenie pierwszej części danych tuż po nawiasie otwierającym (patrz wiersz ❶). Wybierz odpowiednie miejsce, w którym nastąpi przejście do kolejnego wiersza i na końcu bieżącego umieść operator konkatencji (+), tak jak pokazałem w wierszu ❷. Najpierw naciśnij klawisz *Enter*, a później tabulator w celu wyrównania kolejnych wierszy do jednego poziomu wcięć po poleceniu `print()`. Kiedy zakończysz przygotowanie danych wyjściowych, w ostatnim wierszu bloku `print()` umieść nawias zamykający (patrz wiersz ❸).

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 imiona 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, wcięty 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("\nKlucz: " + key) ❷
    print("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 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 (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: last
Wartość: kowalski
```

```
Klucz: first
Wartość: jan
```

```
Klucz: username
Wartość: jkowalski
```

Ponownie zwróć uwagę, że w wyniku iteracji przez słownik pary klucz-wartość nie są zwracane w kolejności ich umieszczenia w słowniku. Python nie przejmuje się kolejnością, w jakiej są przechowywane pary klucz-wartość, a jedynie monitoruje powiązania między poszczególnymi kluczami i ich wartościami.

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("Ulubiony język programowania użytkownika " + name.title() +
        ↪ " to " + ❷
        language.title() + ".")
```

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

```
friends = ['paweł', 'sara'] ❶  
for name in favorite_languages.keys():  
    print(name.title())  
  
    if name in friends: ❷  
        print(" Witaj, " + name.title() +  
              "! Widzę, że Twoim ulubionym językiem programowania jest " +  
              favorite_languages[name].title() + "!") ❸
```

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, wyświetlamy specjalne powitanie odwołujące się do ulubionego języka programowania. Aby uzyskać dostęp do ulubionego języka programowania (patrz wiersz ❸), używamy nazwy słownika oraz bieżącej wartości `name` jako klucza. 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

Słownik zawsze zachowuje jasne połączenie między poszczególnymi kluczami i powiązanimi z nimi wartościami, ale nigdy nie otrzymasz elementów słownika w jakiegokolwiek przewidywalnej kolejności. To nie stanowi problemu, ponieważ zwykle chcesz otrzymać prawidłową wartość powiązaną z danym kluczem.

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': 'ruby',
    'paweł': 'python',
}

for name in sorted(favorite_languages.keys()):
    print(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',  
    'pawel': '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  
Python  
Ruby
```

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.

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.

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.

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("Całkowita liczba obcych: " + str(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:

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

Całkowita liczba obcych: 30

Wprowadzcie 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 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:

```
{'speed': 'średnio', 'color': 'żółty', 'points': 10}
{'speed': 'średnio', 'color': 'żółty', 'points': 10}
```

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

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("Zamówiłeś pizzę na " + pizza['crust'] + " cieście " + ❷
      "wraz z następującymi dodatkami:")

for topping in pizza['toppings']: ❸
    print("\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. 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("\n Ulubione języki programowania użytkownika " + name.title() +
          ↵" to:")
    for language in languages: ❸
        print("\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 Paweł to:
    Python
    Haskell
```

```
Ulubione języki programowania użytkownika Edward to:
    Ruby
    Go
```

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 program. 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("\nNazwa użytkownika: " + username) ❷
    full_name = user_info['first'] + " " + user_info['last'] ❸
    location = user_info['location']

    print("\tImię i nazwisko: " + full_name.title()) ❹
    print("\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. Wprowadzić 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 tworzonej programom, które wreszcie będą mogły reagować na dane wejściowe wprowadzane przez użytkowników.

Skorowidz

A

administrator, 238
adres URL, 513, 524, 531
 aplikacji w Heroku, 587
 dla edit_entry, 539
 dla new_entry, 535
 dla strony rejestracji, 549
 dla strony wylogowania, 547
akceptacja danych wejściowych, 170
aktualny poziom gry, 395
aktywacja
 modeli, 503
 środowiska wirtualnego, 497
alias
 funkcji, 212
 modułu, 213
analiza
 pętli, 90
 tekstu, 267
 wyników, 429
API, 471
aplikacja
 django-bootstrap3, 564
 Learning Log, 496
 Terminal, 37
 users, 543
aplikacje sieciowe, 300, 493
argument, 187
 opcjonalny, 196
 pozycyjny, 188, 190, 207
 w postaci słowa kluczowego, 190
asercja, 284
atak cross-site request forgery, 534
attribut, 219
 errors formularza, 545
 owner, 561

automatyczne obliczanie danych,
 413
automatyczny zapis wykresu, 416
awaria programu, 265

B

baza danych, 499
 SQLite, 583
białe znaki, 57
biblioteka
 matplotlib, 406, 408
 standardowa, 245
bieżący katalog roboczy, 579
blok
 elif, 127
 else, 128, 265
 header, 569
 try-except, 263
błąd
 o kodzie 404, 590
 TypeError, 272
błądzenie losowe, 417, 426
błędy
 indeksu, 86
 logiczne, 95
 serwera, 592
 składni, 59, 60
 typu, 63
 wcięć, 94
Bootstrap, 565
brak
 dwukropka, 97
 wcięcia, 94, 95
budowanie
 mapy świata, 458
 systemu uwierzytelniania, 530

C

ciąg tekstowy, 54
 błędy składni, 59
CSV, comma-separated values, 437
czas epoki systemu UNIX, 479

D

dane wejściowe, 165, 170
data i godzina, 444
definiowanie
 aliasu funkcji, 212
 aliasu modułu, 213
 attributów, 232
 części głównej strony, 569
 funkcji, 185
 krotki, 108
 metod, 232
 modeli, 502
 nagłówków HTML, 566
 paska nawigacji, 567
 własnych kolorów, 415
dekorator @login_required, 552
dezaktywacja przycisku Gra, 376
Django, 495
długość wiersza, 112
docstring, 186
dodawanie
 aplikacji users, 543
 białych znaków, 57
 elementów, 74, 75
 komentarzy, 66
 łącza, 486, 534, 541, 546, 551
 nowych wpisów, 535
 obrazu statku, 314
 pary klucz-wartość, 142

- dodawanie
 - plików do repozytorium, 625
 - podpowiedzi, 483
 - przycisku Gra, 369
 - punktów do wykresu, 424
 - rzędów obcych, 349
 - tematu, 507, 530
 - testu, 288
- dokumentacja
 - biblioteki, 618
 - Pythona, 618
- dołączanie
 - adresów URL, 543
 - do pliku, 261
- dostęp
 - do atrybutów, 221
 - do danych użytkownika, 552
 - do elementów listy, 72
 - do pliku, 252
 - do serwera, 534
 - do słownika, 141
 - do tematów, 558
- dowolna liczba argumentów, 205, 207, 208
- drugie zatwierdzenie, 627
- działanie aplikacji Learning Log, 577
- dziedziczenie, 229, 231
 - szablonu, 518
- dziennik projektu, 626

E

- edycja wpisu, 539, 542
- edytor tekstu, 34, 607
 - Emacs, 614
 - Geany, 36, 43, 608
 - IDLE, 613
 - instalacja, 38, 42
 - Sublime Text, 38, 610
 - vim, 614
- egzemplarz
 - jako atrybut, 234
 - klasy, 217
 - obcego, 342
- element Jumbotron, 570

F

- filtr szablonu, 526
- flaga, 174
- flota obcych, 343, 351
- format

- CSV, 438
- docstring, 214
- JSON, 273, 452
- formatowanie daty i godziny, 444
- formularz, 530
 - modelu, 530
 - modelu dla wpisu, 535
 - UserCreationForm, 550
- framework Django, 495
- funkcja, 185

- check_bullet_alien_collisions(), 386
- check_events(), 317, 327
- close(), 251
- country_code(), 457
- create_fleet(), 348
- edit_entry(), 539
- fire_bullet(), 337, 338
- get_stored_username(), 278
- greet_user(), 277, 279
- input(), 166
- int(), 168
- json.dump(), 273
- json.load(), 273
- logout_view(), 547
- new_entry(), 536
- new_topic(), 531
- open(), 251, 260
- range(), 97, 99
- register(), 549
- rstrip(), 251
- scatter(), 411, 413, 415
- sorted(), 83
- str(), 63
- topic(), 524
- topics(), 521
- update_aliens(), 366
- update_bullets(), 337, 360
- update_screen(), 319

- funkcje
 - alias, 212
 - argument opcjonalny, 196
 - argumenty, 187
 - modyfikowanie listy, 202
 - nadawanie stylu, 214
 - niedopasowane argumenty, 193
 - parametry, 187
 - pętla while, 199
 - przechowywane w modułach, 209
 - przekazywanie argumentów, 205
 - przekazywanie informacji, 186
 - treść, 186

- uniemożliwianie
 - modyfikowania listy, 205
- wartość domyślna, 191
- wartość zwrotna, 195, 198
- wbudowane, 604
- wiele wywołań, 189
- wywołanie, 186

G

- galeria
 - matplotlib, 408
 - Pygal, 427
- generowanie
 - błądeń losowych, 420
 - danych, 405
- Git, 472, 621
 - dodawanie plików, 625
 - drugie zatwierdzenie, 627
 - dziennik projektu, 626
 - ignorowanie plików, 623
 - inicjalizacja repozytorium, 624
 - instalacja, 622
 - przywracanie projektu, 629
 - sprawdzanie stanu, 624
 - stan projektu, 628
 - tworzenie projektu, 623
 - usuwanie repozytorium, 631
 - zatwierdzanie plików, 625
- GitHub, 472
- gra, 303
 - Inwazja obcych, 299, 301, 304
 - statek kosmiczny, 303
- grupowanie państw, 463

H

- Hacker News API, 487
- histogram, 430

I

- identyfikacja użytkowników, 556
- ignorowanie
 - plików, 582, 623
 - wielkości liter, 117
- import
 - całego modułu, 242
 - klas, 237
 - klas z modułu, 241
 - modułu, 210
 - modułu w module, 243
 - określonych funkcji, 212

- wszystkich funkcji modułu, 213
- wszystkich klas z modułu, 242

indeks, 72

informacje o brakujących danych, 451

inicjalizacja repozytorium, 624

inkrementacja wartości atrybutu, 228

instalacja

- edytora tekstu, 34, 38, 42
- frameworka Django, 498
- Gita, 582, 622
 - w systemie Linux, 622
 - w systemie OS X, 622
 - w systemie Windows, 622
- Heroku Toolbelt, 576
- matplotlib, 406
- menedżera pip, 306
- pakietów, 305
- Pygal, 427
- Pygame, 304, 307–309
- Pythona, 40
- requests, 473
- virtualenv, 497
- w systemie Linux, 599
- w systemie OS X, 600
- w systemie Windows, 602

interfejs programowania aplikacji, API, 471

interpreter, 32, 50, 603

Inwazja obcych, 301

iteracja, 89, 109, 150

- przez słownik, 148
- przez wycinek, 104

J

JSON, JavaScript Object Notation, 273

K

kanaly IRC, 618

katalog dla plików statycznych, 580

kierowanie statkiem kosmicznym, 320

kierunek poruszania się floty, 353

klasa, 217

- Alien, 341
- AnonymousSurvey, 293
- Bullet, 330
- Button, 370
- Car, 224

Die, 427

RandomWalk, 417

klasy

- dziedziczenie, 229, 231
- import, 237
- importowane z modułu, 241
- nadawanie stylu, 246
- nadpisywanie metod, 233
- nadrzędne, 233
- potomne, 230, 232
- przechowywane w module, 240
- testowanie, 290
- tworzenie, 218
- użycie, 218

klucze, 140

- zewnątrzne, 508
- słownika, 150
- uporządkowane, 152

kod

- państwa, 455
- wewnątrz pętli, 92

kolejność argumentów, 190

kolizja, 356, 361, 362

kolor tła, 311

kolorowanie punktów, 421

- początkowego i końcowego, 423

komentarze, 65

komponenty gry, 367

konfiguracja

- bazy danych, 585
- edytora Geany, 36, 43
- Gita, 582, 622
- kont użytkowników, 543
- Sublime Text, 38
- superużytkownika, 505

konkatenacja, 56

konstrukcja

- if-elif-else, 125
- if, 115
- if-else, 125

konta użytkowników, 529

konto w Heroku, 576

kontrola wersji, 621

konwencje stylu, 111

konwersja ciągu tekstowego, 454

kopiowanie listy, 105

krotka, 108

- iteracja, 109
- nadpisanie, 110

kursor, 377

L

liczba

- argumentów, 205
- pi, 257
- statków, 396
- wywołań API, 478

liczby

- całkowite, 61, 64
- porządkowe, 136
- zmiennoprzecinkowe, 62

Linux, 33

- instalacja Pythona, 600
- powłoka systemu, 45

lista, 71

- błąd indeksu, 86
- dodawanie elementów, 74, 75
- iteracja, 89
- kopiowanie, 105
- liczbowa, 97
- nieuporządkowana, 522
- określenie wielkości, 84
- pakietów, 577
- polecenie if, 132
- przekazywanie, 200
- pusta, 133
- składana, 101
- słowników, 155
- trwale sortowanie, 82
- tymczasowe sortowanie, 83
- usuwanie elementów, 74, 77–79
- w słowniku, 158
- wartości specjalne, 132
- wstawianie elementów, 76
- wycinek, 103
- wyszukiwanie wartości, 121
- wyświetlanie, 84
- zmienianie elementów, 74

logowanie, 544

lokalne wyświetlanie stron

- błędów, 591

Ł

łącze, 526

- do edit_entry, 541
- do new_entry, 538
- do new_topic, 534
- umożliwiające wylogowanie, 548

łączenie ciągów tekstowych, 55

M

- mapa
 - kolorów, 415
 - populacji, 460
 - świata, 458, 459
- mapowanie
 - adresu URL, 513
 - globalnych zbiorów danych, 452
- menedżer pip, 305
- metoda, 54
 - `__init__()`, 219, 326, 428
 - `count()`, 272
 - `get_object_or_404()`, 592
 - `keys()`, 150
 - `pop()`, 78
 - `remove()`, 81
 - `save()`, 550
 - `setUp()`, 295
 - `sort()`, 82
- metody asercji, 290
- migracja, 500
 - bazy danych, 556
 - modelu Entry, 509
- model
 - Entry, 508
 - Topic, 507, 555
 - rzeczywistych obiektów, 236
- moduł, 185, 210
 - collections, 248
 - datetime, 442, 444
 - game_functions, 317
 - json, 249, 273
 - unittest, 290
- modyfikacja
 - elementów na liście, 75
 - listy, 202
 - modelu Topic, 555
 - pliku base.html, 566
 - pliku settings.py, 579
 - pliku wsgi.py, 580
 - wartości atrybutu bezpośrednia, 226
 - wartości słownika, 143
 - za pomocą metody, 226
- monitorowanie
 - plików projektu, 582
 - wywołań API, 478

N

- nadanie stylu
 - aplikacji, 564, 565
 - dany, 421

- funkcjom, 214
- klasom, 246
- mapie świata, 464
- stronie głównej, 570
- stronie logowania, 570
- stronie new_topic, 571
- stronie tematów, 573
- wpisom, 573
- nadpisywanie
 - krotki, 110
 - metod, 233
- nagłówki HTML, 566
- nakładanie cienia, 447
- narzędzia Bootstrap, 565
- nazwa zmiennej, 51
- nierówność, 118
- niezaliczony test, 285, 286
- notacja wycinka, 205
- numeracja indeksu, 73

O

- obcy, 341
- obiekt, 217
 - screen, 311
- obraz statku, 314
- ochrona
 - bloga, 561
 - strony edit_entry, 560
 - strony new_entry, 561
 - tematów użytkownika, 559
- odczytywanie danych
 - wiersz po wierszu, 253
 - z pliku, 250, 440
- odpowiedniki wywołań funkcji, 192
- ograniczenie
 - dostępu, 552
 - dostępu do strony tematów, 553
 - dostępu w aplikacji, 554
 - liczby pocisków, 336
- opcja SECRET_KEY, 594
- operator
 - `*`, 213
 - modulo, 169
 - równości, 117
- opracowanie specyfikacji, 496
- organizacja listy, 82
- OS X, 37
 - instalacja Homebrew, 601
 - instalacja Pythona, 602
 - powłoka systemu, 45

P

- pakiet
 - Pygal, 426, 479
 - requests, 473
 - virtualenv, 497
- panel, 573
- para klucz-wartość, 140, 142
- parametr, 187
 - linewidth, 410
- pasek nawigacji, 567
- pętla
 - for, 91, 93
 - nieskończona, 177
 - while, 171, 179, 199
 - zdarzeń, 311
- pierwszy program, 32
- pip, 305
- planowanie projektu, 304
- plik
 - .gitignore, 582
 - 404.html, 590
 - admin.py, 506, 509
 - alice.py, 266
 - alien.py, 140, 341, 352
 - alien_invasion.py, 310, 328, 342, 345, 353, 356, 367
 - aliens.py, 155
 - americas.py, 458
 - amusement_park.py, 126
 - apostrophe.py, 59
 - banned_users.py, 122
 - bar_descriptions.py, 483
 - base.html, 518, 523, 545, 548, 566, 569
 - bicycles.py, 72
 - birthday.py, 63
 - bullet.py, 328–331
 - button.py, 370
 - car.py, 224, 237, 240
 - cars.py, 82, 116
 - cities.py, 175
 - comment.py, 65
 - confirmed_users.py, 179
 - counting.py, 171, 176, 177
 - countries.py, 456
 - country_codes.py, 456
 - dice_visual.py, 431
 - die.py, 427
 - die_visual.py, 428–430
 - different_dice.py, 433
 - dimensions.py, 109
 - division.py, 262, 264

dog.py, 218
electric_car.py, 230, 243
even_or_odd.py, 170
favorite_languages.py, 146,
149, 159, 245
file_reader.py, 250, 254
foods.py, 105
formatted_name.py, 195
forms.py, 530, 535
game_functions.py, 318, 321,
322, 324, 328, 329, 333, 336,
337, 338, 340, 343, 346, 348,
349, 353, 355, 357, 359, 360,
361, 364, 366, 367, 372, 373,
376, 377, 380, 383, 385, 394
game_stats.py, 363, 367, 370,
389
greet_user.py, 275
greet_users.py, 201
greeter.py, 167, 186, 199
highs_lows.py, 438, 440, 446,
450
hn_submissions.py, 488
index.html, 516, 519, 570
language_survey.py, 291
login.html, 545
magic_number.py, 119
magicians.py, 90, 94
making_pizzas.py, 211
many_users.py, 161
models.py, 502, 508, 555
mountain_poll.py, 181
mpl_squares.py, 408–410
my_car.py, 239
my_cars.py, 241, 242
my_electric_car.py, 241
na_populations.py, 459
name.py, 54
name_function.py, 282, 285, 287
names.py, 282
new_entry.html, 537
new_topic.html, 533, 572
number_reader.py, 274
number_writer.py, 273
parrot.py, 166, 172
person.py, 198
pets.py, 180, 188
pi_digits.txt, 250
pi_string.py, 255, 257
pizza.py, 159, 206, 211
placeholder.txt, 581
players.py, 103
population_data.json, 453
printing_models.py, 202

Proffie, 580
programming.txt, 260
python_repos.py, 474–480, 486
random_walk.py, 418
register.html, 551
remember_me.py, 275, 276, 277
requirements.txt, 577
runtime.txt, 578
rw_visual.py, 419, 423, 425
scatter_squares.py, 412, 413
scoreboard.py, 382, 388, 390,
393, 397
settings.py, 313, 325, 329, 336,
352, 378, 379, 387, 503, 543,
553, 579, 588
ship.py, 314, 322–325, 327,
329, 365, 396
squares.py, 100, 101
survey.py, 291
test_name_function.py, 284
test_survey.py, 293
topic.html, 525, 538, 541, 574
topics.html, 522, 526, 534
toppings.py, 118, 129, 132
urls.py, 513, 520, 524, 531,
536, 539, 544, 547, 549
user_profile.py, 208
views.py, 515, 521, 524, 531,
536, 540, 548, 549, 553, 560,
593
voting.py, 124
word_count.py, 269
world_population.py, 453–457,
460, 463, 464
write_message.py, 259, 261
wsgi.py, 580
pliki, 249
.py, 583
.pyc, 583
CSV, 438
odczytywanie danych, 250
ogromne, 257
statyczne, 576, 580
ścieżka dostępu, 252
tryb dołączania, 261
tworzenie listy wierszy, 255
zapisywanie danych, 258
pobranie
danych, 437
dwuznakowego kodu państwa,
455
pocisk, 330
podklasy, 231
podpowiedź, 483, 485

polecenia if-else, 125
polecenie
break, 175
continue, 176
del, 77
if, 123, 132
import this, 68
python, 33, 39
pominięcie bloku else, 128
pomoc, 615
poprawianie wykresu, 410
populacja świata, 452, 465
porównania liczbowe, 119
poruszanie
flotą obcych, 351
statkiem, 323
potrójny cudzysłów, 186
powiązanie
danych, 555
tematu z użytkownikiem, 560
powierzchnia, 311
powłoka, 37
uruchamianie programów, 45
Django, 510
poziom
gry, 391
trudności, 377
problem z instalacją, 44
projekt, 299
gry, 310
w Django, 498
w GitHub, 481
przechowywanie
danych, 273
funkcji w modułach, 209
pocisków, 332
przegląd projektu, 340, 500
przekazywanie
argumentów, 188
listy, 200
projektu do Heroku, 584
zmian, 592
przerwanie pętli, 175
przestrzeń nazw, 518
przesuwanie
floty, 355
obcych, 352
przetwarzanie
nagłówków pliku CSV, 438
odpowiedzi API, 473
przycisk Gra, 369, 376
przywracanie
projektu, 629
stanu projektu, 628

przyznawanie dostępu, 558
punktacja, 369, 381
puste wiersze, 113
Python, 27
Python User Group, PUG, 598

R

reakcja
 na działania użytkownika, 310
 na kolizję, 362
refaktoryzacja, 277, 278, 317, 426, 561
 funkcji `check_events()`, 327
 funkcji `create_fleet()`, 348
 funkcji `update_bullets()`, 360
rejestracja
 modelu, 506
 modelu `Entry`, 509
 użytkownika, 548, 551
repozytorium, 472, 477, 624
 usunięcie, 631
rodzaje
 komentarzy, 66
 metod asercji, 290
rozbudowa aplikacji, 593
rozjaśnianie motywu graficznego, 467
równość, 116
rzut
 dwoma kośćmi, 431
 kością do gry, 428
 kośćmi o różnej liczbie ścianek, 433

S

seria danych, 446, 448
serwer `gunicorn`, 581
serwis `GitHub`, 472
skrót klawiaturowy
 `Command+Spacja`, 37
 `Ctrl+B`, 39
 `Ctrl+D`, 34
słownik, 140, 245
 iteracja, 148
 klucze, 150
 klucze uporządkowane, 152
 modyfikowanie wartości, 143
 odpowiedzi, 474
 para klucz-wartość, 140
 podobne obiekty, 145
 pusty, 143
 w słowniku, 161

słowo kluczowe, 604
 `and`, 120
 `as`, 212
 `or`, 121
 `username`, 186
sortowanie listy, 82, 83
specyfikacja
 PEP 8, 112
 projektu, 496
 stylu, 113
sprawdzanie
 dziennika projektu, 626
 równości, 117
 stanu projektu, 624
 wielu warunków, 120, 121, 128
 menedżera `pip`, 305, 306
 nierówności, 118
 równości, 116
 wersji, 33
stan projektu, 624, 628
statek kosmiczny, 303, 314
 dodanie obrazu, 314
 dostosowanie szybkości, 325
 kierowanie, 320
 ograniczenie zasięgu, 327
 umożliwienie nieustannego ruchu, 321
 wyrzeliwanie pocisków, 330, 333
 wyświetlenie, 316
 zmiana kierunku, 323
statystyka listy liczb, 101
stos wywołań, 44, 52, 262
strona
 główna aplikacji, 512
 logowania, 544, 546, 572
 `new_entry`, 539
 `new_topic`, 531
 rejestracji użytkownika, 548, 551
 tematów, 520
strony
 błędu, 590
 poszczególnych tematów, 523
styl
 aplikacji `Learning Log`, 564
 funkcji, 214
 klasy, 246
 kodu, 111
 mapy świata, 464
 poleceń `if`, 136
 strony głównej, 570
 strony logowania, 570
 strony `new_topic`, 571

 strony tematów, 573
 wpisów, 573
style `Pygal`, 465
superklasy, 231
superużytkownik, 505, 530
 w `Heroku`, 586
symulacja rzutu kością, 426
system
 kontroli wersji, 472, 621
 operacyjny, 33
 operacyjny `Linux`, 33
 operacyjny `OS`, 37
 operacyjny `Windows`, 40
 punktacji, 400
szablon, 512, 516
 `Bootstrap`, 565
 dla `edit_entry`, 540
 dla `new_entry`, 537
 dla strony logowania, 545
 dla strony `new_topic`, 533
 dla strony rejestracji, 551
 nadrzędny, 518
 potomny, 519
 tematów, 522, 525
 znacznika, 518
szybkość pocisku, 360

Ś

ścieżka dostępu do pliku
 bezwzględna, 253
 względna, 252
środowisko
 lokalne, 581
 programistyczne, 31
 uruchomieniowe `Pythona`, 578
 wirtualne, 496, 497

T

tablica wyników, 383
test
 jednostkowy, 283
 warunkowy, 116, 122
testowanie
 funkcji, 282
 klasy, 290
 klasy `AnonymousSurvey`, 293
 kodu, 281
tło, 311
tryb
 dołączania, 259, 261
 odczytu, 259
 zapisu, 259

tworzenie
bazy danych, 499
dodatkowych stron, 517
egzemplarza, 217
egzemplarza na podstawie klasy, 220
egzemplarza obcego, 342
floty, 358
floty obcych, 343
gry, 299, 303
histogramu, 430
klasy, 218, 220
 Alien, 341
 Bullet, 330
 Button, 370
 Die, 427
 RandomWalk, 417
 statku kosmicznego, 314
 ustawień, 312
komentarzy, 65
konta w Heroku, 576
list liczbowych, 97
listy pakietów, 577
listy wierszy, 255
obcego, 341
tworzenie okna Pygame, 310
pliku Procfie, 580
projektu, 623
rzędów obcych, 345
stron błędu, 590
stron internetowych, 512
superużytkownika, 586
szablону, 516, 590
środowiska wirtualnego, 496
tablicy wyników, 383
widoku, 515
wielu egzemplarzy, 222
wizualizacji, 477

U

uaktualnienie punktacji, 384
ukrycie
 kursora myszy, 377
 osi, 423
unikanie
 awarii programu, 264
 błędów składni, 59
 błędów typu, 63
uprawnienia, 238, 505
uruchamianie
 komponentów gry, 367
 procesu, 580

programów z poziomu powłoki, 45
aplikacji, 501
gry, 373
programu, 34, 39, 44
Pythona, 40
usuwanie
 elementu, 74, 77
 metoda pop(), 78
 na podstawie wartości, 80
 polecenie del, 77
 z dowolnego miejsca na liście, 79
 z listy, 77
białych znaków, 57
konturów, 415
niewidocznych pocisków, 334
pary klucz-wartość, 145
projektu z Heroku, 594
repozytorium, 631
uwierzytelnianie, 530
uwzględnienie trafień obcych, 386
użycie
 bloku try-except, 263
 bloków elif, 127
 Bootstrapa, 565
 elementu Jumbotron, 570
 flagi, 174
 funkcji int(), 168
 funkcji range(), 99
 Gita, 582, 621
 klasy, 218
 mapy kolorów, 415
 pętli while, 179
 polecenia continue, 176
 serwera gunicorn, 581
 słowa kluczowego and, 120
 słowa kluczowego or, 121
 strony logowania, 546
 wartości listy, 73
 Web API, 471
 wielu list, 134
 zmiennych, 51, 52

W

wartość, 50
 boolowska, 122
 domyślna, 191
 domyślna atrybutu, 225
 zwrotna, 195
wcięcie, 94, 112
 brak, 95
 niepotrzebne, 95

wczytywanie całego pliku, 250
wdrażanie
 aplikacji Learning Log, 575
 projektu, 585
Web API, 471
widok, 512, 515
 tematów, 521, 524
wielkość populacji, 461
wiersz poleceń
 uruchomienie Pythona, 40
Windows, 40
 instalacja Pythona, 602
 wiersz poleceń, 46
witryna
 administracyjna, 507
 administracyjna Django, 505
 Hacker News, 487
wizualizacja
 danych, 299, 403
 Pygal, 426
 repozytoriów, 479
własne
 strony błędu, 590
 szablony, 590
wprowadzanie danych, 530
wstawianie elementów, 76
wybór kierunku kroku, 418
wycinek listy, 103
wyjątek, 249, 262
 FileNotFoundError, 266
 ZeroDivisionError, 262
wykres
 automatyczny zapis, 416
 błądzenia losowego, 419
 dodawanie łączy, 486
 dodawanie punktów, 424
 generowany przez Pygal, 481
 liniowy, 408, 443
 kolorowanie punktów, 415
 modyfikowanie, 410
 nakładanie cienia, 447
 punktowy, 413
 serie danych, 448
 słupkowy, 431
 symulacji rzucania kośćmi, 433
 temperatury, 442
 typu World, 460
 ukrywanie osi, 423
 wyświetlanie punktów, 411
 wyświetlanie serii punktów, 413
 zmienianie etykiety, 409
 zmienianie grubości, 409
 zmienianie wielkości, 425

- wykrywanie kolizji, 356, 361
- wylogowanie, 547
- wyodrębnienie danych, 440, 453
- wyrażenie boolowskie, 122
- wystrzeliwanie pocisków, 330, 333
- wyszukiwanie informacji, 617
- wyświetlanie
 - danych, 60, 484
 - danych liczbowych, 459
 - danych na wykresie, 442
 - daty, 444
 - drugiej serii danych, 446
 - liczby statków, 396
 - listy, 84
 - mapy populacji, 460
 - nagłówków, 439
 - obcego, 343
 - populacji, 466
 - poziomu gry, 391
 - punktacji, 381
 - przedziału czasu, 445
 - przycisku, 372
 - serii punktów, 413
 - statku kosmicznego, 316
 - wykresu błędzenia losowego, 419

- wywołanie
 - API, 472
 - print(), 93
 - metody, 221
- wyzerowanie szybkości, 380
- wzorzec adresu URL, 520, 524, 526

Z

- zabezpieczenie wdrożonego projektu, 588
- zagnieżdżanie, 154
- zakończenie
 - działania programu, 172
 - gry, 361
- zaliczenie testu, 283
- zaokrąglanie punktacji, 388
- zapisywanie
 - danych w pliku, 258
 - wielu wierszy, 260
- zapytania, 525
- zatwierdzenie, commit, 582
- plików, 625
- projektu, 583
- zmian, 589
- zawartość pliku, 255
- zbiór, 154
- zdarzenie, 311

- zdobywanie punktów, 387
- Zen Pythona, 66
- zerowanie gry, 374
- zestaw testów, 283
 - o pełnym pokryciu, 283
- zestrzeliwanie obcych, 356, 358
- zgłaszanie błędów, 271
- zmiana
 - elementów, 74
 - kierunku floty, 355
 - poziomu trudności, 377
 - szybkości, 378
 - szybkości pocisku, 360
 - wielkości liter, 54
 - wielkości wykresu, 425
- zmienna Path, 603
- zmiennie, 50
- znacznik <div>, 569
- znak
 - gwiazdki, 206, 214
 - nowego wiersza, 57
 - pionowej linii, 526
- zwrot słownika, 198

Ż

- żądanie
 - danych, 472
 - GET, 532
 - POST, 532

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

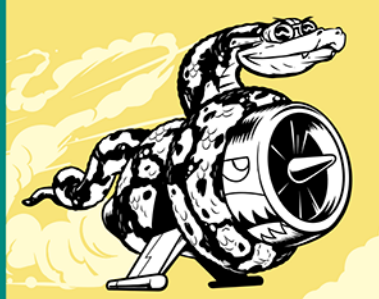
Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**



PRZEKONAJ SIĘ, JAK SZYBKO MOŻESZ ZACZAĆ
TWORZYĆ ŚWIETNE APLIKACJE W PYTHONIE!

Python to język programowania o bardzo wszechstronnych możliwościach. Nadaje się do tworzenia gier i aplikacji sieciowych oraz wdrażania indywidualnych rozwiązań biznesowych. Wykorzystuje się go do różnych celów naukowych i budowania praktycznych rozwiązań najrozmaitszych problemów. Umożliwia pisanie przejrzystego, zwięzłego kodu, który jest łatwy w konserwacji i pozwala na sprawne rozwijanie oprogramowania. Jeśli chcesz zacząć szybko pisać funkcjonujący, efektywny kod i tworzyć działające aplikacje, Python jest świetnym wyborem.

Niniejsza książka jest zwięzłym, praktycznym podręcznikiem programowania w Pythonie. Dzięki niej gruntownie opanujesz podstawy języka i nabierzesz dobrych nawyków w programowaniu. Szybko będziesz mógł skoncentrować się na praktycznej stronie realizacji projektów, a nowo poznane koncepcje wypróbujesz przez opracowywanie konkretnych kwestii. W ten sposób przygotujesz się do nauki zaawansowanych technik Pythona.

Najważniejsze zagadnienia omówione w książce:

- podstawowe koncepcje programowania
- praktyczne sposoby obsługi błędów i testowania kodu
- biblioteki i narzędzia Pythona, takie jak matplotlib, NumPy i Pygal
- praca z danymi i generowanie interaktywnych wizualizacji
- tworzenie praktycznych aplikacji od podstaw i ich wdrażanie na serwerach WWW
- możliwe problemy i sposoby ich rozwiązywania

Eric Matthes — uczy fizyki i matematyki w szkole średniej. Od kilku lat prowadzi kursy dla początkujących programistów Pythona. Swój pierwszy program komputerowy — prostą, poprawnie funkcjonującą grę — napisał w wieku 5 lat. Mieszka na Alasce z żoną i synem.

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

Helion

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

ISBN 978-83-283-2595-1



9 788328 325951

cena: 89,00 zł

