

O'REILLY®

Wydanie II

Python

Nowoczesne programowanie
w prostych krokach



Helion 

Bill Lubanovic

Tytuł oryginału: *Introducing Python: Modern Computing in Simple Packages*, 2nd Edition

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-6842-2

© 2020 Helion SA

Authorized Polish translation of the English edition of *Introducing Python 2E*

ISBN 9781492051367 © 2020 Bill Lubanovic

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

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 Helion SA 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 Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/pythno>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/pythno.zip>

Printed in Poland.

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

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

Wstęp	21
--------------------	-----------

Część I. Podstawy Pythona

1. Przedsmak Pythona	29
Zagadki	29
Kilka małych programów	31
Większy program	33
Python w praktyce	36
Python i inne języki	37
Dlaczego właśnie Python?	39
Kiedy nie używać Pythona?	41
Wersje Pythona 2 i 3	42
Instalacja Pythona	42
Uruchomienie Pythona	42
Interaktywny interpreter	43
Skrypty	43
Co dalej?	44
Twoja chwila zen	44
Już wkrótce	45
Do zrobienia	45
2. Dane — typy, wartości, zmienne i nazwy	47
Dane w Pythonie to obiekty	47
Typy danych	48
Mutowalność	49
Wartości	49
Zmienne	50
Przypisania	51

Zmienne to nazwy, a nie miejsca w pamięci	52
Przypisania wielokrotne	55
Ponowne przypisania	55
Kopiowanie	55
Nadawaj zmiennym właściwe nazwy	56
Już wkrótce	57
Do zrobienia	57
3. Liczby	59
Typ logiczny	59
Typ całkowity	60
Literały	60
Działania	61
Zmienne	63
Kolejność działań	65
Podstawa systemu liczbowego	65
Konwersja typów	68
Jak duża może być liczba całkowita?	69
Typ zmiennoprzecinkowy	70
Funkcje matematyczne	71
Już wkrótce	71
Do zrobienia	72
4. Podejmowanie decyzji, czyli instrukcja if	73
Komentowanie za pomocą znaku #	73
Kontynuacja wiersza za pomocą znaku \	74
Sprawdzanie za pomocą instrukcji if, elif i else	75
Co oznacza True?	77
Wielokrotne sprawdzanie za pomocą operatora in	78
Nowość: operator walrus	79
Już wkrótce	80
Do zrobienia	80
5. Ciągi znaków	81
Tworzenie ciągu znaków za pomocą cudzysłowów i apostrofów	81
Tworzenie ciągu za pomocą funkcji str()	84
Ucieczka ze znakiem \	84
Łączenie ciągów za pomocą znaku +	85
Powielanie ciągu za pomocą znaku *	86
Wyodrębnianie znaku za pomocą nawiasów []	86
Wyodrębnianie fragmentu ciągu za pomocą wycinka	87
Określanie długości ciągu za pomocą funkcji len()	89

Dzielenie ciągu za pomocą funkcji split()	89
Łączenie ciągów za pomocą funkcji join()	90
Zastępowanie fragmentów ciągu za pomocą funkcji replace()	90
Przycinanie ciągu za pomocą funkcji strip()	91
Przeszukiwanie i wybieranie ciągów znaków	92
Wielkości liter	93
Wyrównywanie ciągu znaków	94
Formatowanie ciągów znaków	94
Formatowanie w starym stylu za pomocą znaku %	94
Formatowanie w starym stylu za pomocą znaków {} i funkcji format()	97
Najnowszy styl formatowania: f-ciągi	98
Więcej o ciągach	99
Już wkrótce	99
Do zrobienia	100
6. Pętle while i for	101
Powtarzanie operacji za pomocą instrukcji while	101
Przerywanie pętli za pomocą instrukcji break	102
Powrót na początek pętli za pomocą instrukcji continue	102
Sprawdzanie przerwania pętli za pomocą instrukcji else	103
Iterowanie za pomocą instrukcji for i in	103
Przerywanie pętli za pomocą instrukcji break	104
Powrót na początek pętli za pomocą instrukcji continue	104
Sprawdzanie przerwania pętli za pomocą instrukcji else	104
Generowanie sekwencji liczb za pomocą funkcji range()	105
Inne iteratory	106
Już wkrótce	106
Do zrobienia	106
7. Krotki i listy	107
Krotki	107
Tworzenie krotek za pomocą przecinków i nawiasów	107
Tworzenie krotek za pomocą funkcji tuple()	109
Łączenie krotek za pomocą znaku dodawania	109
Powielanie elementów krotki za pomocą znaku mnożenia	109
Porównywanie krotek	109
Iterowanie krotek za pomocą instrukcji for i in	109
Modyfikowanie krotek	110
Listy	110
Tworzenie list za pomocą nawiasów kwadratowych	110
Tworzenie i przekształcanie list za pomocą funkcji list()	111

Tworzenie list z ciągów za pomocą funkcji split()	111
Odczytywanie elementu listy za pomocą nawiasów kwadratowych i indeksu	112
Wyodrębnianie elementów listy za pomocą wycinków	112
Dołączanie elementu do listy za pomocą funkcji append()	113
Wstawianie elementu wewnątrz listy za pomocą funkcji insert()	113
Powielanie elementów listy za pomocą znaku mnożenia	114
Łączenie list za pomocą funkcji extend() i znaku dodawania	114
Modyfikowanie elementu listy za pomocą nawiasów kwadratowych i indeksu	114
Modyfikowanie elementów listy za pomocą wycinka	115
Usuwanie elementu listy za pomocą instrukcji del i indeksu	115
Usuwanie elementu o zadanej wartości za pomocą funkcji remove()	116
Odczytywanie i usuwanie elementu listy za pomocą funkcji pop()	116
Usuwanie wszystkich elementów listy za pomocą funkcji clear()	116
Wyszukiwanie elementu o zadanej wartości za pomocą funkcji index()	117
Sprawdzanie zawartości listy za pomocą instrukcji in	117
Zliczanie wystąpień wartości za pomocą funkcji count()	117
Przekształcanie listy w ciąg znaków za pomocą funkcji join()	117
Zmienianie kolejności elementów za pomocą funkcji sort() i sorted()	118
Sprawdzanie długości listy za pomocą funkcji len()	119
Przypisywanie listy zmiennej za pomocą znaku równości	119
Kopiowanie listy za pomocą funkcji copy() i list() oraz wycinka	119
Kopiowanie zawartości listy za pomocą funkcji deepcopy()	120
Porównywanie list	121
Iterowanie listy za pomocą instrukcji for i in	121
Iterowanie kilku list za pomocą funkcji zip()	122
Tworzenie listy za pomocą wyrażenia listowego	123
Listy list	125
Krotki a listy	126
Nie ma wyrażen krotkowych	126
Już wkrótce	126
Do zrobienia	127
8. Słowniki i zbiory	129
Słowniki	129
Tworzenie słownika za pomocą nawiasów klamrowych	129
Tworzenie słownika za pomocą funkcji dict()	130
Przekształcanie struktury w słownik za pomocą funkcji dict()	130
Dodawanie i zmienianie elementów słownika za pomocą nawiasów kwadratowych i klucza	131
Odczytywanie wartości za pomocą nawiasów kwadratowych i klucza oraz funkcji get()	132

Odczytywanie wszystkich kluczy za pomocą funkcji keys()	133
Odczytywanie wszystkich wartości za pomocą funkcji values()	133
Odczytywanie wszystkich par klucz-wartość za pomocą funkcji items()	133
Odczytywanie długości słownika za pomocą funkcji len()	133
Łączenie słowników za pomocą nawiasów klamrowych i znaków mnożenia	134
Łączenie słowników za pomocą funkcji update()	134
Usuwanie elementu słownika za pomocą instrukcji del i klucza	135
Odczytywanie i usuwanie elementu słownika za pomocą funkcji pop()	135
Usuwanie wszystkich elementów słownika za pomocą funkcji clear()	135
Sprawdzanie dostępności klucza za pomocą instrukcji in	136
Przypisywanie wartości za pomocą znaku równości	136
Kopiowanie słownika za pomocą funkcji copy()	136
Kopiowanie zawartości słownika za pomocą funkcji deepcopy()	137
Porównywanie słowników	138
Iterowanie elementów słownika za pomocą instrukcji for i in	138
Wyrażenia słownikowe	139
Zbiory	140
Tworzenie zbioru za pomocą funkcji set()	140
Przekształcanie struktury w zbiór za pomocą funkcji set()	141
Odczytywanie długości zbioru za pomocą funkcji len()	141
Dodawanie elementów do zbioru za pomocą funkcji add()	141
Usuwanie elementów zbioru za pomocą funkcji remove()	142
Iterowanie zbioru za pomocą instrukcji for i in	142
Sprawdzanie dostępności elementu za pomocą instrukcji in	142
Kombinacje wartości i operatory	143
Wyrażenia zbiorowe	145
Tworzenie niemutowalnych zbiorów za pomocą funkcji frozenset()	146
Poznane struktury danych	146
Tworzenie większych struktur	147
Już wkrótce	148
Do zrobienia	148
9. Funkcje	149
Definiowanie funkcji za pomocą instrukcji def	149
Wywołanie funkcji z nawiasami	150
Argumenty i parametry	150
Wartość None jest przydatna	152
Argumenty pozycyjne	153
Argumenty nazwane	153
Domyślne wartości parametrów	154
Rozbijanie/zbieranie argumentów pozycyjnych za pomocą znaku *	155

Rozbijanie/zbieranie argumentów nazwanych za pomocą znaków **	156
Stosowanie wyłącznie argumentów nazwanych	157
Argumenty mutowalne i niemutowalne	158
Komentarze dokumentacyjne	158
Funkcje jako typy pierwszoklasowe	159
Funkcje wewnętrzne	161
Domknięcia	161
Funkcje anonimowe: wyrażenia lambda	162
Generatory	163
Funkcje generatorowe	163
Wyrażenia generatorowe	164
Dekoratory	164
Przestrzenie nazw i zakresy widoczności	166
Podwójne znaki podkreślenia w nazwach	168
Rekurencja	169
Funkcje asynchroniczne	170
Wyjątki	170
Obsługiwanie wyjątków za pomocą instrukcji try i except	171
Definiowanie własnych wyjątków	172
Już wkrótce	173
Do zrobienia	173
10. Obiekty i klasy	175
Czym jest obiekt?	175
Proste obiekty	176
Definiowanie klasy	176
Atrybuty	177
Metody	178
Inicjacja obiektu	178
Dziedziczenie klas	179
Dziedziczenie klasy nadrzędnej	179
Nadpisywanie metod	181
Dodawanie metod	182
Odwołanie do klasy nadrzędnej za pomocą metody super()	182
Dziedziczenie wielu klas	183
Domieszki	185
Argument self	185
Dostęp do atrybutów	186
Dostęp bezpośredni	186
Gettery i settery	186
Dostęp do atrybutów za pośrednictwem właściwości	187

Właściwości zwracające wyliczane wartości	188
Prywatność dzięki modyfikacji nazwy	189
Atrybuty klas i obiektów	190
Rodzaje metod	191
Metody instancji	191
Metody klasy	191
Metody statyczne	192
Kacze typowanie	192
Magiczne metody	194
Agregowanie i komponowanie klas	196
Kiedy stosować obiekty, a kiedy inne struktury	197
Nazwane krotki	198
Klasy danych	199
Pakiet attrs	200
Już wkrótce	201
Do zrobienia	201
11. Moduły, pakiety i inne rzeczy	203
Moduły i instrukcja import	203
Importowanie modułu	203
Importowanie modułu z przemianowaniem	205
Importowanie tylko tego, co jest potrzebne	205
Pakiety	206
Ścieżka wyszukiwania modułów	207
Importowanie względne i bezwzględne	208
Przestrzenie pakietów	208
Moduły a obiekty	209
Przydatne rzeczy w standardowej bibliotece Pythona	210
Obsługa brakujących kluczy za pomocą funkcji setdefault() i defaultdict()	210
Zliczanie elementów za pomocą klasy Counter	212
Porządkowanie elementów według klucza za pomocą klasy OrderedDict	213
Stos + kolejka == deque	213
Iterowanie struktur danych za pomocą modułu itertools	214
Czytelne wyświetlanie za pomocą funkcji pprint()	215
Wartości losowe	216
Więcej narzędzi: zewnętrzny kod	217
Już wkrótce	217
Do zrobienia	217

Część II. Python w praktyce

12. Żonglowanie danymi	221
Dane tekstowe — kodowanie Unicode	222
Kodowanie Unicode w Pythonie 3	222
Schemat UTF-8	225
Kodowanie znaków	225
Dekodowanie znaków	227
Kody HTML	228
Normalizowanie znaków	229
Dodatkowe informacje	230
Dane tekstowe — wyrażenia regularne	230
Wyszukiwanie dokładnego początkowego dopasowania za pomocą funkcji match()	231
Wyszukiwanie pierwszego dopasowania za pomocą funkcji search()	233
Wyszukiwanie wszystkich dopasowań za pomocą funkcji findall()	233
Dzielenie ciągu według dopasowań za pomocą funkcji split()	233
Zastępowanie dopasowań za pomocą funkcji sub()	234
Wzorce: znaki specjalne	234
Wzorce: specyfikatory	235
Wzorce: określanie formatu wyniku funkcji	238
Dane binarne	238
Struktury bytes i bytearray	238
Konwertowanie danych binarnych za pomocą modułu struct	240
Inne narzędzia do przetwarzania danych binarnych	242
Konwertowanie bajtów i ciągów znaków za pomocą modułu binascii	243
Operatory bitowe	243
Już wkrótce	244
Do zrobienia	244
13. Data i czas	247
Rok przestępny	248
Moduł datetime	248
Moduł time	251
Odczytywanie i wyświetlanie daty i czasu	253
Wszystkie przekształcenia	256
Alternatywne moduły	257
Już wkrótce	257
Do zrobienia	257

14. Pliki i katalogi	259
Zapisywanie i odczytywanie plików	259
Tworzenie i otwieranie plików za pomocą funkcji open()	259
Zapisywanie danych tekstowych za pomocą instrukcji print()	260
Zapisywanie danych tekstowych za pomocą instrukcji write()	261
Odczytywanie pliku tekstowego za pomocą funkcji read(), readline() i readlines()	262
Zapisywanie danych binarnych za pomocą funkcji write()	264
Odczytywanie danych binarnych za pomocą funkcji read()	264
Automatyczne zamykanie pliku za pomocą instrukcji with	265
Przesuwanie wskaźnika za pomocą funkcji seek()	265
Mapowanie pamięci	267
Operacje na plikach	267
Sprawdzanie dostępności pliku za pomocą funkcji exists()	267
Sprawdzanie rodzaju elementu za pomocą funkcji isfile()	267
Kopiowanie pliku za pomocą funkcji copy()	268
Zmianianie nazwy pliku za pomocą funkcji rename()	268
Tworzenie odnośników za pomocą funkcji link() i symlink()	268
Zmianianie uprawnień do pliku za pomocą funkcji chmod()	269
Zmianianie właściciela pliku za pomocą funkcji chown()	269
Usuwanie pliku za pomocą funkcji remove()	269
Operacje na katalogach	270
Tworzenie katalogu za pomocą funkcji mkdir()	270
Usuwanie katalogu za pomocą funkcji rmdir()	270
Odczytywanie zawartości katalogu za pomocą funkcji listdir()	270
Zmianianie katalogu za pomocą funkcji chdir()	271
Wyszukiwanie plików i katalogów za pomocą funkcji glob()	271
Nazwy ścieżek	271
Uzyskiwanie ścieżki za pomocą funkcji abspath()	272
Uzyskiwanie ścieżki na podstawie symbolicznego odnośnika za pomocą funkcji realpath()	272
Tworzenie ścieżki za pomocą funkcji os.path.join()	272
Moduł pathlib	273
Klasy BytesIO i StringIO	274
Już wkrótce	275
Do zrobienia	275
15. Dane w czasie, czyli procesy i współbieżność	277
Programy i procesy	277
Tworzenie procesu za pomocą modułu subprocess	278
Tworzenie procesu za pomocą modułu multiprocessing	279
Przerywanie procesu za pomocą funkcji terminate()	280

Uzyskiwanie informacji systemowych za pomocą modułu os	280
Uzyskiwanie informacji o procesie za pomocą pakietu psutil	281
Automatyzacja poleceń	282
Pakiet invoke	282
Inne przydatne pakiety	283
Współbieżność	283
Kolejki	284
Procesy	285
Wątki	285
Moduł concurrent.futures	288
Zielone wątki i biblioteka gevent	290
Moduł twisted	293
Biblioteka asyncio	294
Biblioteka Redis	294
Nie tylko kolejki	297
Już wkrótce	298
Do zrobienia	298
16. Dane w pudełku, czyli trwale zapisywanie	299
Płaskie pliki tekstowe	299
Dopełniane pliki tekstowe	299
Tabelaryczne pliki tekstowe	300
Format CSV	300
Format XML	302
Dane XML i bezpieczeństwo systemu	304
Format HTML	305
Format JSON	305
Format YAML	308
Moduł tablib	309
Biblioteka Pandas	309
Pliki konfiguracyjne	310
Pliki binarne	311
Dopełniane pliki binarne i mapowanie pamięci	311
Arkusze kalkulacyjne	312
Format HDF5	312
Format TileDB	312
Relacyjne bazy danych	312
Język SQL	313
Interfejs DB-API	315
Baza SQLite	315
Baza MySQL	317

Baza PostgreSQL	317
Biblioteka SQLAlchemy	318
Inne pakiety do obsługi baz danych	323
Bazy danych NoSQL	323
Rodzina formatów dbm	323
Baza Memcached	324
Baza Redis	325
Bazy dokumentowe	331
Bazy danych czasowych	332
Bazy grafowe	332
Inne bazy NoSQL	333
Bazy pełnotekstowe	333
Już wkrótce	334
Do zrobienia	334
17. Dane w przestrzeni, czyli sieć	335
Protokoły sieciowe	335
Gniazda sieciowe	336
Narzędzie Scapy	340
Narzędzie Netcat	341
Zasady komunikacji sieciowej	341
Zasada „zapytanie-odpowiedź”	341
Biblioteka ZeroMQ	342
Inne narzędzia do przesyłania komunikatów	345
Zasada „publikuj-subskrybuj”	346
Serwer Redis	346
Biblioteka ZeroMQ	347
Inne narzędzia typu „publikuj-subskrybuj”	349
Usługi internetowe	349
Usługa DNS	349
Moduły do obsługi poczty e-mail	350
Inne protokoły	350
Usługi WWW i interfejsy API	351
Serializacja danych	351
Moduł pickle	352
Inne formaty serializacyjne	353
Protokół RPC	353
Protokół XML-RPC	354
Protokół JSON-RPC	355
Biblioteka MessagePack RPC	356

Biblioteka zerorpc	357
Biblioteka gRPC	357
Biblioteka Twirp	358
Narzędzia do zdalnego zarządzania	358
Big Fat Data	359
Hadoop	359
Spark	359
Disco	360
Dask	360
Chmury	360
Amazon Web Services	361
Google Cloud	362
Microsoft Azure	362
OpenStack	362
Docker	362
Kubernetes	362
Już wkrótce	363
Do zrobienia	363
18. Sieć WWW	365
Klienci internetowi	366
Testowanie strony za pomocą programu telnet	367
Testowanie strony za pomocą programu curl	368
Testowanie strony za pomocą programu httpie	369
Testowanie strony za pomocą programu httpbin	369
Standardowe biblioteki WWW w Pythonie	369
Nie tylko standardowa biblioteka: moduł requests	371
Serwery WWW	373
Najprostszy serwer WWW	373
Interfejs WSGI	374
Interfejs ASGI	375
Serwer Apache	375
Serwer NGINX	376
Inne serwery WWW dla języka Python	377
Platformy dla serwerów WWW	377
Platforma Bottle	378
Platforma Flask	380
Platforma Django	384
Inne platformy	384

Platformy bazodanowe	385
Usługi WWW i automatyzacja	385
Moduł webbrowser	386
Moduł webview	386
Interfejsy API i REST	387
Automatyczne wyszukiwanie danych	388
Scrapy	388
BeautifulSoup	388
Moduł requests-html	389
Obejrzyjmy film	390
Już wkrótce	391
Do zrobienia	392
19. Stań się pythonistą	393
O programowaniu	393
Skąd wziąć kod?	394
Instalowanie pakietów	394
Program pip	395
Program virtualenv	395
Program pipenv	396
Menedżer pakietów	396
Instalacja plików źródłowych	396
Środowiska IDE	397
IDLE	397
PyCharm	397
IPython	398
Jupyter Notebook	399
JupyterLab	399
Nazwy i dokumentacja	399
Wskazówki typów	401
Testy	401
Programy pylint, pyflakes, flake8 i pep8	402
Pakiet unittest	403
Pakiet doctest	407
Pakiet nose	408
Inne platformy testujące	409
Ciągła integracja	409
Diagnostyka kodu	410
Funkcja print()	410
Dekoratory	411
Moduł pdb	412
Funkcja breakpoint()	417

Rejestrowanie komunikatów o błędach	417
Optymalizacja kodu	419
Pomiary czasu	419
Algorytmy i struktury danych	422
Cython, NumPy i rozszerzenia C	423
PyPy	424
Numba	424
Kontrola wersji kodu	425
Mercurial	425
Git	425
Rozpowszechnianie programów	428
Dowiedz się więcej	428
Książki	428
Strony internetowe	429
Grupy	429
Konferencje	429
Praca	430
Już wkrótce	430
Do zrobienia	430
20. Python w sztuce	431
Grafika dwuwymiarowa	431
Standardowa biblioteka	431
Biblioteki PIL i Pillow	432
ImageMagick	435
Grafika trójwymiarowa	436
Animacja trójwymiarowa	436
Graficzne interfejsy użytkownika	436
Wykresy i wizualizacja danych	438
Matplotlib	438
Seaborn	440
Bokeh	441
Gry	442
Dźwięk i muzyka	442
Już wkrótce	443
Do zrobienia	443
21. Python w biznesie	445
Pakiety biurowe	445
Operacje biznesowe	446

Przetwarzanie danych biznesowych	447
Wyodrębnianie, przekształcanie i ładowanie danych	447
Weryfikacja poprawności danych	450
Dodatkowe źródła informacji	451
Otwarte pakiety biznesowe	451
Python i finanse	451
Bezpieczeństwo danych finansowych	452
Mapy	452
Formaty danych	453
Wyświetlanie map w formacie shapefile	453
Geopandas	455
Inne pakiety kartograficzne	457
Aplikacje i dane	458
Już wkrótce	459
Do zrobienia	459
22. Python w nauce	461
Matematyka i statystyka w standardowej bibliotece	461
Funkcje matematyczne	461
Liczby zespolone	463
Precyzja obliczeń i moduł decimal	464
Działania na ułamkach zwykłych	464
Klasa array, czyli spakowana sekwencja wartości	465
Prosta statystyka w module statistics	465
Mnożenie macierzy	465
Naukowy Python	465
Pakiet NumPy	466
Tworzenie tablicy za pomocą funkcji array()	467
Tworzenie tablicy za pomocą funkcji arange()	467
Tworzenie tablicy za pomocą funkcji zeros(), ones() i random()	468
Zmienianie kształtu tablicy za pomocą metody reshape()	469
Odwoływanie się do elementów tablicy za pomocą nawiasów []	469
Działania na tablicach	470
Algebra liniowa	471
Biblioteka SciPy	472
Biblioteka SciKit	472
Biblioteka Pandas	473
Python w różnych dziedzinach nauki	473
Już wkrótce	474
Do zrobienia	474

A Sprzęt i oprogramowanie dla początkującego programisty	475
B Instalacja Pythona 3	485
C Coś z zupełnie innej beczki: asynchroniczność	493
D Rozwiązania zadań	499
E Ściągawka	539

Przedsmak Pythona

Uznanie zyskują tylko okropne języki. Python jest wyjątkiem.

— Don Knuth

Zagadki

Zacznijmy od dwóch minizagadek i ich rozwiązań. Jak myślisz, co oznaczają poniższe wiersze?

(Rząd 1): (PS) K18, pomiń, k1, odwróć.

(Rząd 2): (LS) S1 1 dług., p5, 2o. na lewo, p1, odwróć.

Zapisy wyglądają technicznie, niczym jakiś program komputerowy. W rzeczywistości jest to *wzór dziewiarski* na piętę skarpety, takiej jak na rysunku 1.1.



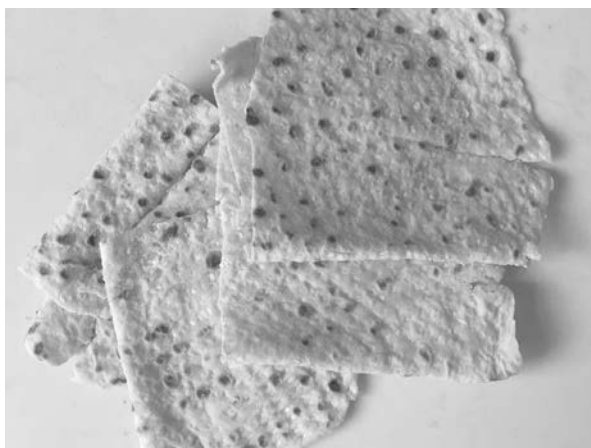
Rysunek 1.1. Skarpety zrobione na drutach

Dla mnie ten wzór jest równie zrozumiały jak łamigłówka sudoku dla moich kotów. Natomiast dla mojej żony jest oczywisty. Dla Ciebie pewnie też, jeżeli lubisz robotki ręczne.

Przyjrzyjmy się innemu tajemniczemu tekstowi — liście składników. Jej przeznaczenie jest zupełnie jasne, choć nie wiadomo, jaki jest końcowy produkt.

1/2 k. masła lub margaryny
1/2 szk. kremu
2 1/2 szk. mąki
1 mała łyż. soli
1 duża łyż. cukru
4 szk. startych ziemniaków (zimnych)
Wszystkie składniki przed zmieszaniem z mąką muszą być zimne.
Wymieszaj wszystkie składniki.
Dokładnie ugnieć.
Uformuj 20 kulek i umieść je w chłodnym miejscu.
Dla każdej kulki:
Rozsyp mąkę na szmatce.
Rozwałkuj kulkę na okrągły placek.
Smaż na patelni do uzyskania brązowego koloru.
Odwróć i smaż na drugiej stronie.

Nawet jeżeli nie gotujesz, poznasz, że jest to przepis kulinarny¹. Składa się z listy składników i wskazuje do wykonania. Ale na co to jest przepis? Na lefse, norweski specjał, podobny do tortilli (patrz rysunek 1.2). Posmaruj go masłem, dżemem lub czymkolwiek innym, zwiń i smacznego!



Rysunek 1.2. Lefse

Wzór dziewiarski i przepis kulinarny mają kilka wspólnych cech:

- Ustalony **zbiór** słów, skrótów i symboli. Niektóre są zrozumiałe, inne brzmią tajemniczo.
- **Składnię**, czyli zasady określające sposób wyrażania treści.
- **Sekwencję operacji**, które należy wykonać w zadanej kolejności.

¹ Zazwyczaj można go znaleźć w książkach kucharskich lub tajemnych zapiskach kucharzy.

- Sekwencje powtarzanych operacji (**pętle**), na przykład sposób smażenia placka lefse.
- Odwołania do innych sekwencji operacji (**funkcji** w terminologii informatycznej). W przepisie kulinarnym może to być odwołanie do przepisu ścierania ziemniaków.
- Założenie dotyczące znajomości **kontekstu**. W przepisie przyjęte jest założenie, że ten, kto go czyta, wie, co to jest woda i jak się ją gotuje. Wzór dziewiarski zakłada, że umiesz robić na drutach i nie kłujesz się nimi zbyt często.
- Wykorzystywane, tworzone i modyfikowane **dane** — tutaj ziemniaki i włóczka.
- **Narzędzia** do przetwarzania danych — garnek, mikser, patelnia, druty.
- Oczekiwany efekt. W opisanych przykładach jest to coś, co można zjeść lub na siebie założyć.

Tego rodzaju przykłady możesz znaleźć wokół siebie wszędzie. Są to idiomy, żargony lub specjalne języki. Ludziom, którzy je znają, pozwalają oszczędzić czas, ale dla innych brzmią tajemniczo. Spróbuj zrozumieć artykuł o brydżu, jeżeli w niego nie grasz, lub specjalistyczną publikację, jeżeli nie jesteś naukowcem (lub nim jesteś, tylko w innej dziedzinie).

Kilka małych programów

Wszystkie opisane w poprzednim podrozdziale elementy znajdziesz w programach komputerowych, napisanych w specyficznych językach, za pomocą których ludzie mówią komputerom, co mają robić. Użyłem przykładów ze wzorem dziewiarskim i przepisem kulinarnym, aby pokazać, że w programowaniu nie ma niczego tajemniczego. Opanowanie tej sztuki polega głównie na zapamiętaniu pewnych słów i zasad ich stosowania.

Jest dobrze, jeżeli nie ma zbyt wielu słów i zasad, których trzeba się nauczyć. Ludzki mózg jest w stanie pomieścić tylko określoną ilość informacji.

Zajmijmy się wreszcie jakimś prostym programem (patrz listing 1.1). Jak myślisz, co on robi?

Listing 1.1. Program odliczanie.py

```
for countdown in 5, 4, 3, 2, 1, "Cześć!":
    print(countdown)
```

Jeżeli domyśliłeś się, że jest to program w języku Python wyświetlający następujące wiersze:

```
5
4
3
2
1
Cześć!
```

przyznasz, że łatwiej jest zrozumieć kod napisany w tym języku niż przepis kulinarny czy wzór dziewiarski. Poza tym programy w Pythonie możesz pisać w komfortowym i bezpiecznym zacisku swojego gabinetu, z dala od gorących garnków i ostrych przedmiotów.

Programowanie w Pythonie polega na stosowaniu specjalnych słów i symboli (`for`, `in`, `print`, przecinków, średników, nawiasów), będących ważnymi elementami składni (zasad) tego języka. Dobra wiadomość jest taka, że składnia Pythona jest przyjemniejsza i łatwiejsza do opanowania niż w innych językach. Kod napisany w tym języku brzmi całkiem naturalnie — prawie jak przepis kulinarny.

Listing 1.2 przedstawia inny program, który wybiera z listy jedno z zaklęć Harry’ego Pottera i wyświetla je na ekranie.

Listing 1.2. Program zaklęcia.py

```
spells = [  
    "Riddikulus!",  
    "Wingardium Leviosa!",  
    "Avada Kedavra!",  
    "Expecto Patronum!",  
    "Nox!",  
    "Lumos!",  
]  
print(spells[3])
```

Poszczególne zaklęcia są **ciągami znaków** (sekwencjami znaków ujętymi w cudzysłowy). Rozdzielone przecinkami i ujęte w nawiasy klamrowe ([i]), tworzą **listę**. Słowo spells jest **zmienną**, czyli nazwą, za pomocą której można z listą robić różne rzeczy. W tym przykładzie program wyświetla czwarte zaklęcie.

```
Expecto Patronum!
```

Dlaczego do wyświetlenia czwartego zaklęcia została użyta cyfra 3? Lista w Pythonie jest sekwencją wartości, do których można odwoływać się za pomocą **przesunięcia** względem początku listy. Pierwsza wartość ma przesunięcie równe 0, a czwarta równe 3.



Ludzie odliczają od 1, zatem odliczanie od zera wygląda przedziwnie. Dlatego lepiej jest używać pojęcia *przesunięcie* zamiast *pozycja*. Jest to jeden z przykładów różnic pomiędzy językiem programowania a językiem ludzi.

Lista jest bardzo popularną **strukturą danych** w Pythonie. W rozdziale 7. dowiesz się, jak ją stosować.

Program przedstawiony w listingu 1.3 wyświetla jeden z cytatów z „Kubusia Puchatka”. Tym razem jednak użyte jest odwołanie wykorzystujące nazwę bohatera, a nie pozycję w liście.

Listing 1.3. Program cytaty.py

```
quotes = {  
    "Puchatek": "Jestem Misiem o Bardzo Małym Rozumku",  
    "Kłapouchy": "Wypadek to bardzo dziwna rzecz. Nigdy go nie ma, dopóki się nie wydarzy.",  
    "Tygrysek": "Jest to tak skomplikowane, że nawet ja to rozumiem.",  
}  
character = "Tygrysek"  
print(character, "powiedział:", quotes[character])
```

Po uruchomieniu tego prostego programu na ekranie pojawi się:

```
Tygrysek powiedział: Jest to tak skomplikowane, że nawet ja to rozumiem.
```

Słowo quotes jest zmienną, w tym przypadku nazwą **słownika**. Słownik to kolekcja unikatowych kluczy (tutaj imion bohaterów) i skojarzonych z nimi wartości (cytatów). Często jest to bardziej przydatna struktura niż lista, ponieważ umożliwia odwoływanie się do wartości za pomocą nazw.

W listingu 1.2 zostały użyte nawiasy kwadratowe ([i]) do zdefiniowania listy, a w listingu 1.3 nawiasy klamrowe ({ i }) do zdefiniowania słownika. Klucze z wartościami w słowniku są kojarzone za pomocą dwukropka (:). Więcej o słownikach dowiesz się w rozdziale 8.

Mam nadzieję, że nie była to dla Ciebie zbyt duża dawka informacji o składni. W kolejnych rozdziałach po trochu będziesz dowiadywał się coraz więcej o obowiązujących prostych regułach.

Większy program

Teraz zajmiemy się czymś zupełnie innym. Listing 1.4 przedstawia program wykonujący serię dość skomplikowanych operacji. Na razie nie staraj się zrozumieć, jak ten program działa — ta książka jest od tego, abyś w odpowiednim czasie się tego dowiedział. Celem przykładu jest pokazanie Ci, jak wygląda typowy niebanalny program w Pythonie. Jeżeli znasz inne języki programowania, możesz je porównać z Pythonem. Nie znasz wprawdzie jeszcze tego języka, ale czy bez zaglądania do opisu niżej potrafisz dociec, co robią poszczególne wiersze? Poznałeś już przykłady użycia listy i słownika. Tutaj pojawia się kilka nowości.

Listing 1.4. Program archive.py

```
1 import webbrowser
2 import json
3 from urllib.request import urlopen
4
5 print("Poszukajmy jakiejś starej strony.")
6 site = input("Podaj adres URL strony: ")
7 era = input("Podaj rok, miesiąc i dzień, np. 20150613: ")
8 url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
9 response = urlopen(url)
10 contents = response.read()
11 text = contents.decode("utf-8")
12 data = json.loads(text)
13 try:
14     old_site = data["archived_snapshots"]["closest"]["url"]
15     print("Adres znalezionej kopii: ", old_site)
16     print("Za chwilę strona powinna pojawić się w przeglądarce.")
17     webbrowser.open(old_site)
18 except:
19     print("Niestety, tej strony nie ma w archiwum.")
```

Przykładowy program opisany w poprzednim wydaniu książki łączył się z witryną YouTube. Pobierał informacje o najwyższej ocenianych filmach i robił to dobrze. Ledwie jednak wyschła farba drugiego wydania, Google zablokowało wykorzystywaną w programie funkcję, więc przestał on działać. Nowy przykład, przedstawiony w listingu 1.4, odwołuje się do innej witryny, dzięki której będzie działał nieco dłużej: Internet Archive (<http://archive.org>). Jest to bezpłatny serwis, w którym przechowywane są miliardy stron (jak również filmy, muzyka, gry i inne cyfrowe starocie), które pojawiły się w Internecie w ciągu ostatnich 20 lat. Więcej przykładów użycia interfejsu API udostępnianego w tej witrynie poznasz w rozdziale 18.

Program prosi użytkownika o podanie adresu URL strony i daty. Następnie wysyła do witryny Internet Archive zapytanie, czy jest dostępna kopia żądanej strony z zadanego okresu. Jeżeli tak, program wyświetla jej adres w archiwum i otwiera ją w przeglądarce. Celem przykładu jest pokazanie, jak za pomocą Pythona można wykonywać różnego rodzaju operacje, takie jak pobieranie danych od użytkownika, przesyłanie ich przez Internet, odczytywanie odebranych treści, wyodrębnianie adresów URL i wyświetlanie stron w przeglądarce.

Po odebraniu z Internetu strony zapisanej w formacie HTML trzeba ją w jakiś sposób wyświetlić. Jest z tym mnóstwo roboty, którą spokojnie można scedować na przeglądarkę. Można też próbować wyodrębnić z kodu potrzebne dane (więcej informacji o surfowaniu po Internecie znajdziesz w rozdziale 18.). Każda z tych opcji wymaga włożenia dużo pracy w napisanie większego programu. Na szczęście witryna Internet Archive zwraca dane w czytelnym dla człowieka formacie JSON (ang. *JavaScript Object Notation* — zapis obiektów JavaScript), opisującym w uporządkowany sposób typy danych i ich zawartość. Format ten jest kolejnym językiem powszechnie stosowanym do przesyłania informacji pomiędzy różnymi komputerami i systemami. Więcej na jego temat dowiesz się w rozdziale 12.

W Pythonie można przekształcać dane JSON w struktury typowe dla tego języka. Tego rodzaju operacje poznasz w dalszych rozdziałach. Ten prosty program wyodrębnia tylko część danych — adres URL strony zapisanej w archiwum. Niemniej jednak jest to kompletny program, który możesz uruchomić. Aby kod był prosty, zaimplementowany w nim został tylko podstawowy mechanizm obsługi błędów. Numery wierszy nie stanowią części programu — są wykorzystywane jedynie w umieszczonym pod nim opisie.

Ten prosty, składający się z kilkunastu wierszy program robi bardzo wiele rzeczy. Użyte w nim instrukcje mogą nie być jeszcze dla Ciebie zrozumiałe, ale poznasz je w najbliższych rozdziałach. Poniżej znajduje się opis operacji wykonywanych w poszczególnych wierszach.

1. *Importowanie* (udostępnianie w programie) całego kodu modułu `webbrowser` zawartego w standardowej bibliotece Pythona.
2. Importowanie całego kodu modułu `json` zawartego w standardowej bibliotece Pythona.
3. Importowanie wyłącznie funkcji `urlopen()` znajdującej się w module `urllib.request`, zawartym w standardowej bibliotece Pythona.
4. Pusty wiersz, aby program był bardziej czytelny.
5. Wyświetlenie powitalnego tekstu na ekranie.
6. Wyświetlenie prośby o podanie adresu URL, odczytanie danych wpisanych przez użytkownika i zapisanie ich w zmiennej o nazwie `s`.
7. Wyświetlenie następnej prośby, o podanie roku, miesiąca i dnia, a następnie zapisanie wpisanych danych w zmiennej o nazwie `era`.
8. Utworzenie zmiennej o nazwie `url` i umieszczenie w niej danych, których witryna Internet Archive potrzebuje do wyszukania podanej strony.
9. Nawiązanie połączenia z usługą i zażądanie informacji o podanym adresie URL.
10. Odebranie odpowiedzi i przypisanie jej do zmiennej.
11. Zdekodowanie treści zapisanej w formacie JSON i przypisanie jej do zmiennej.
12. Przekształcenie tekstu na dane (strukturę).
13. Obsługa błędów: jeżeli podczas wykonywania kolejnych czterech wierszy wystąpi błąd, nastąpi przejście do ostatniego wiersza programu (poniżej instrukcji `except`).
14. Jeżeli zostanie znaleziona strona z zadanego okresu, nastąpi wyodrębnienie jej adresu z trójpoziomowego słownika. Zwróć uwagę, że ten wiersz i dwa następne są wcięte. W ten sposób oznacza się wiersze, które są objęte instrukcją `try`.

15. Wyświetlenie adresu URL żądanej strony w archiwum.
16. Wyświetlenie informacji o tym, co się za chwilę stanie.
17. Otwarcie w przeglądarce strony o znalezionym adresie URL.
18. Jeżeli podczas wykonywania czterech poprzednich wierszy wystąpi błąd, nastąpi przejście do tego wiersza.
19. Wyświetlenie komunikatu, że żądana strona nie została znaleziona. Wiersz jest wcięty, ponieważ ma być wykonany tylko wtedy, gdy zostanie wykonana poprzedzająca go instrukcja except.

Po uruchomieniu programu oraz wpisaniu adresu URL i daty pojawi się pokazany niżej przykładowy wynik:

```
$ python archive.py
```

Poszukajmy jakiejś starej strony.

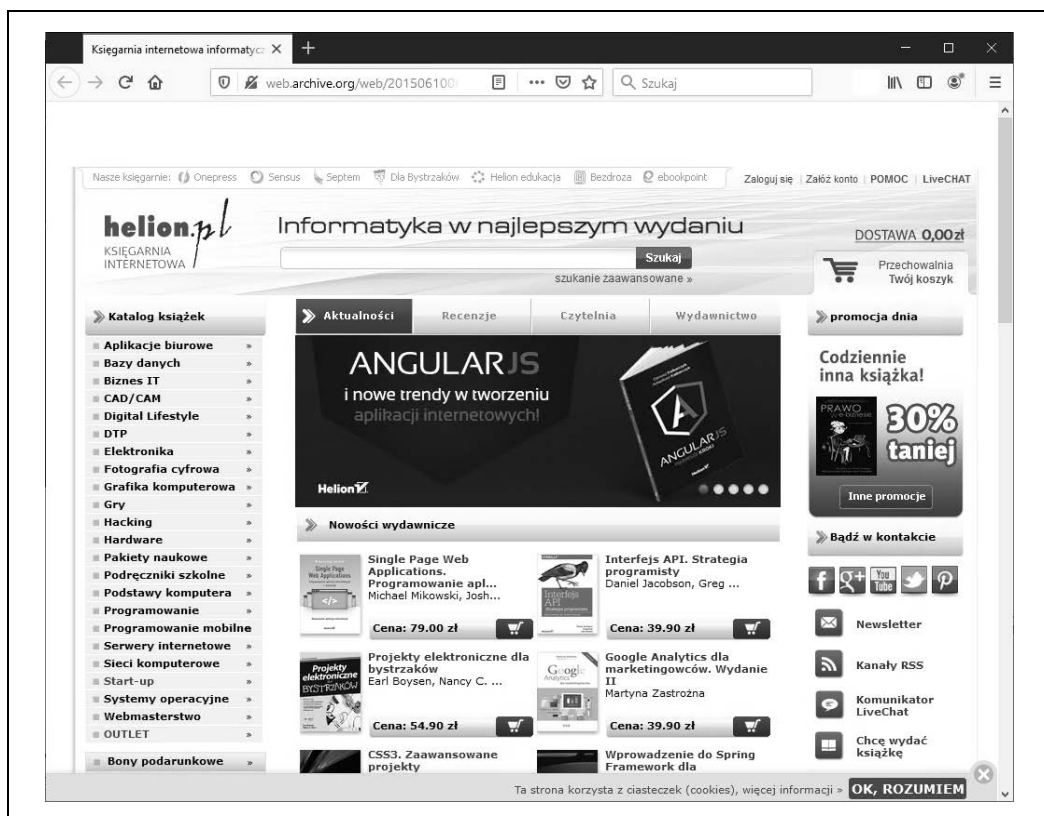
Podaj adres URL strony: **helion.pl**

Podaj rok, miesiąc i dzień, np. 20150613: **20150613**

Adres znalezionej kopii: <http://web.archive.org/web/20150610064343/http://helion.pl/>

Za chwilę strona powinna pojawić się w przeglądarce.

Rysunek 1.3. przedstawia znaną stronę wyświetloną w przeglądarce.



Rysunek 1.3. Strona znaleziona w witrynie Internet Archive

W opisanym przykładzie wykorzystanych jest kilka modułów (programów zainstalowanych razem z Pythonem) ze **standardowej biblioteki**. Jednak nie jesteś skazany na korzystanie wyłącznie z nich. Dostępnych jest wiele doskonałych zewnętrznych modułów. Listing 1.5 przedstawia inną wersję programu, w której wykorzystany jest doskonały zewnętrzny pakiet requests.

Listing 1.5. Program *archive2.py*

```
1 import webbrowser
2 import requests
3
4 print("Poszukajmy jakiejś starej strony.")
5 site = input("Podaj adres URL strony: ")
6 era = input("Podaj rok, miesiąc i dzień, np. 20150613: ")
7 url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
8 response = requests.get(url)
9 data = response.json()
10 try:
11     old_site = data["archived_snapshots"]["closest"]["url"]
12     print("Adres znalezionej kopii: ", old_site)
13     print("Za chwilę strona powinna pojawić się w przeglądarce.")
14     webbrowser.open(old_site)
15 except:
16     print("Niestety, tej strony nie ma w archiwum.")
```

Nowa wersja programu jest krótsza i dla większości programistów bardziej czytelna. Więcej o module requests dowiesz się w rozdziale 18., a o zewnętrznych modułach w ogólności — w rozdziale 11.

Python w praktyce

Czy warto poświęcać czas i siły na uczenie się Pythona? Ten język istnieje od 1991 r. (jest starszy od Javy, ale młodszy od C) i wciąż należy do pięciu najbardziej popularnych języków programowania. Programiści zarabiają pieniądze, tworząc w tym języku poważne serwisy, takie jak Google, YouTube, Instagram, Netflix czy Hulu. Sam używam go do tworzenia produkcyjnych aplikacji wykorzystywanych w różnych dziedzinach. Python ma opinię wydajnego języka, atrakcyjnego dla szybko rozwijających się firm.

Python jest stosowany w różnych obszarach, m.in.:

- wierszach poleceń i terminalach,
- interfejsach graficznych,
- serwisach internetowych po stronie serwera i klienta,
- serwerach wspomagających duże, popularne witryny internetowe,
- chmurach (centrach danych zarządzanych przez zewnętrzne podmioty),
- urządzeniach przenośnych,
- systemach wbudowanych.

Spektrum tworzonych w tym języku programów jest bardzo szerokie — od kilkuwierszowych skryptów, takich jak powyższe przykłady, do potężnych systemów złożonych z milionów linii kodu.

Na stronie <https://oreil.ly/8vK7y> dostępny jest raport Python Developers Survey 2018, zawierający wiele liczb i wykresów pokazujących miejsce Pythona w dzisiejszej informatyce.

Z tej książki dowiesz się, jak ten język jest wykorzystywany w witrynach internetowych oraz w systemach przetwarzania danych i zarządzania nimi. W końcowych rozdziałach opisane są zastosowania Pythona w sztuce, nauce i biznesie.

Python i inne języki

Jak wypada Python w porównaniu z innymi językami? Gdzie i kiedy należy go stosować? W tym podrozdziale przedstawię przykłady napisane w innych językach, abyś miał wyobrażenie, jak wygląda konkurencja. *Nie* wymagam, abyś je rozumiał, szczególnie jeżeli nie miałeś z nimi do czynienia. (Nim dotrzesz do ostatniego przykładu w tej książce, odetchniesz z ulgą, że nie musiałeś pracować z innymi językami).

Każdy z programów wyświetla liczbę i krótką informację o sobie.

Program używany w terminalu lub wierszu poleceń, odczytujący wpisywane przez użytkownika informacje, uruchamiający inne programy i wyświetlający wyniki, jest nazywany **powłoką**. W systemie Windows jest to program `cmd` (<https://pl.wikipedia.org/wiki/Cmd.exe>). Służy on do uruchamiania plików wsadowych z rozszerzeniem `.bat`. W systemach Linux i Unix (w tym macOS) dostępnych jest wiele różnych powłok. Najpopularniejsza z nich to `bash` (<https://www.gnu.org/software/bash/>), zwana też `sh`. Możliwości powłoki są dość ograniczone — pozwalają kodować proste programy i zastępować nazwy plików gwiazdką. Polecenia przeznaczone do późniejszego wykorzystania można zapisywać w plikach zwanych **skryptami powłoki**. Być może były to pierwsze programy, z jakimi miałeś do czynienia jako programista. Problem ze skryptami polega na tym, że nie skalują się dobrze i trudno jest napisać program składający się z więcej niż kilkuset wierszy. Poza tym skrypty działają wolniej niż programy napisane w innych językach. Poniższy listing przedstawia prosty skrypt powłoki.

```
#!/bin/sh
language=0
echo "Język nr $language: cześć, jestem powłoką!"
```

Gdy zapiszesz powyższy kod w pliku `test.sh` i uruchomisz go, na ekranie pojawi się następująca informacja:

```
Język 0: cześć, jestem powłoką!
```

Języki wiarusy, takie jak C ([https://pl.wikipedia.org/wiki/C_\(język_programowania\)](https://pl.wikipedia.org/wiki/C_(język_programowania))) i C++ (<https://pl.wikipedia.org/wiki/C%2B%2B>), to języki niskopoziomowe, których używa się wtedy, gdy ważna jest szybkość programu. Twój system operacyjny i spora liczba dostarczanych z nim programów (włącznie z interpreterem Pythona) prawdopodobnie są napisane w jednym z tych języków.

Języki C i C++ są dość skomplikowane, a napisane w nich programy — trudne w utrzymaniu. Kodując w nich, trzeba pamiętać o wielu szczegółach, m.in. o **zarządzaniu pamięcią**. Pomyłki skutkują trudnymi w diagnozowaniu awariami i błędami. Poniżej przedstawiony jest prosty program napisany w języku C:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int language = 1;
    printf("Język nr %d: cześć, jestem C!", language);
    return 0;
}
```

Język C++ jest podobny do C, ale ma kilka bardziej zaawansowanych funkcjonalności. Poniżej przedstawiony jest przykładowy kod.

```
using namespace std;
int main() {
    int language = 2;
    cout << "Język nr " << language << ": cześć jestem C++!";
    return(0);
}
```

Java (<https://www.java.com>) i C# (<https://oreil.ly/1wo5A>) to następcy C i C++, wolni od wielu mankamentów swoich poprzedników, m.in. skomplikowanego zarządzania pamięcią. Tworzone programy są jednak dość rozbudowane. Przykład kodu napisanego w języku Java wygląda tak:

```
public class Anecdote {
    public static void main (String[] args) {
        int language = 3;
        System.out.format("Język nr %d: cześć, jestem Java!", language);
    }
}
```

Jeżeli nie programowałeś w żadnym z tych języków, możesz się zastanawiać, do czego to wszystko jest potrzebne w programie, który wyświetla jeden wiersz tekstu. W niektórych językach narzut składniowy jest bardzo duży. Więcej na ten temat dowiesz się w rozdziale 2.

Języki C, C++ i Java to przykłady **języków statycznych**, które wymagają określania pewnych niskopoziomowych szczegółów, takich jak typy danych. W dodatku A dowiesz się, że na przykład liczba całkowita zajmuje w pamięci komputera określoną liczbę bitów i na danych tego rodzaju można wykonywać tylko operacje arytmetyczne. Przeciwnieństwem są **języki dynamiczne** (zwane **językami skryptowymi**), w których nie jest wymagane deklarowanie typów zmiennych przed ich użyciem.

Przez wiele lat popularnym, uniwersalnym językiem dynamicznym był Perl (<http://www.perl.org>). Jest to potężny język, wyposażony w wiele przydatnych bibliotek. Jego składnia jest jednak dość kłopotliwa i dlatego w ostatnich latach stracił uznanie na rzecz języków Python i Ruby. Oto jak wygląda nasz przykład napisany w Perlu:

```
my $language = 4;
print "Język nr $language: cześć jestem Perl!";
```

Bardziej nowoczesnym językiem jest Ruby (<http://www.ruby-lang.org>), który odziedziczył kilka cech po Perlu i zdobył popularność dzięki platformie Ruby on Rails, przeznaczonej do tworzenia stron internetowych. Jest wykorzystywany w tych samych obszarach co Python. Wybór między tymi dwoma językami zazwyczaj jest kwestią gustu i dostępności bibliotek ułatwiających stworzenie potrzebnej aplikacji. Poniżej przedstawiony jest przykład napisany w języku Ruby:

```
language = 5
puts "Język nr #{language}: cześć, jestem Ruby!"
```

Użyty w poniższym przykładzie język PHP (<http://www.php.net>) jest bardzo popularny w dziedzinie stron internetowych, ponieważ można go łatwo łączyć z kodem HTML. Sam w sobie ma jednak kilka niedostatków i poza Internetem nie ma wielu zastosowań. Przykład jego użycia wygląda jak niżej:

```
<?PHP
$language = 6;
echo "Język nr $language: cześć, jestem PHP!";
?>
```

W ostatnim czasie pojawił się język Go (lub Golang, gdybyś chciał poszukać o nim informacji w Google), sprawiający wrażenie języka wydajnego i przyjaznego (<https://golang.org>):

```
package main
import "fmt"
func main() {
    language := 7
    fmt.Printf("Język nr %d: cześć, jestem Go!", language)
}
```

Współczesną alternatywą dla C i C++ jest Rust (<https://doc.rust-lang.org>):

```
fn main() {
    let language = 8;
    println!("Język nr {}: cześć, jestem Rust!", language)
}
```

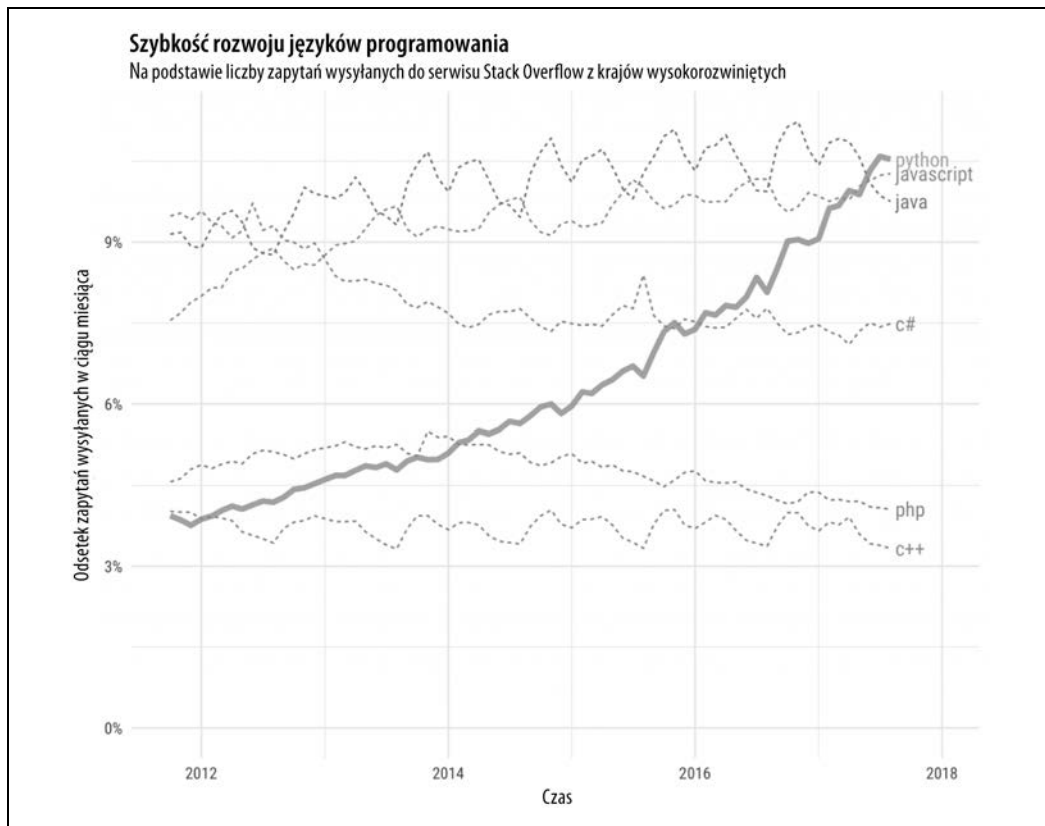
Co jeszcze nam zostało? Ależ tak, Python (<https://python.org>):

```
language = 9
print(f"Język nr {language}: cześć, jestem Python!")
```

Dlaczego właśnie Python?

Jednym, choć nie najważniejszym, powodem, dla którego warto używać Pythona, jest jego popularność. Według różnych źródeł jest to:

- Jeden z najszybciej rozwijających się języków programowania (<https://oreil.ly/YHqqD>), co ilustruje rysunek 1.4.
- Najbardziej rozwojowy język programowania. W czerwcu 2019 r. autorzy indeksu TIOBE (<https://www.tiobe.com/tiobe-index>) podali, że „w tym miesiącu indeks Pythona pobił rekord wszech czasów i osiągnął wartość 8,5%. Jeżeli ten trend się utrzyma, Python w ciągu 3 – 4 lat może zastąpić języki C i Java i stać się najpopularniejszym językiem programowania na świecie”.



Rysunek 1.4. Pod względem szybkości rozwoju Python jest liderem wśród języków programowania

- Najpopularniejszy język programowania w 2018 r. według serwisów TIOBE, IEEE Spectrum (<https://oreil.ly/saRgb>) i PyPL (<http://pypl.github.io/PYPL.html>).
- Najczęściej wybierany język na wykładach z podstaw programowania na największych amerykańskich uczelniach (<http://bit.ly/popular-py>).
- Oficjalnie zatwierdzony w szkołach wyższych we Francji język do nauki programowania.

W ostatnich latach Python zyskał ogromną popularność w dziedzinach badania danych i uczenia maszynowego. Jeżeli chcesz zdobyć dobrze płatną pracę programisty w ciekawej branży, Python obecnie wydaje się najlepszym wyborem. Jeżeli jesteś pracodawcą, możesz wybierać spośród rosnącej liczby doświadczonych programistów.

Dlaczego ten język jest taki popularny? Czy są jakieś ukryte powody?

Python jest uniwersalnym, wysokopoziomowym językiem programowania. Dzięki swojej strukturze jest bardzo *czytelny*, co jest ważniejsze, niż się na pozór wydaje. Każdy program komputerowy pisany jest tylko raz, ale czytany i weryfikowany wielokrotnie, często przez różne osoby. Czytelność języka decyduje o łatwości nauczania się go i zapamiętania. Krzywa uczenia Pythona w porównaniu z innymi językami jest łagodna, co oznacza, że szybko można go zacząć skutecznie wykorzystywać, ale głębokie poznanie wymaga czasu i doświadczenia.

Dzięki zwięzłości Pythona programy są krótsze od napisanych w innych językach. Badania wykazały, że programiści, niezależnie od używanego języka, piszą dziennie mniej więcej stałą liczbę wierszy kodu. Zatem im mniej wierszy musi użyć programista do osiągnięcia jakiegoś celu, tym większa jest jego wydajność. Dla wielu firm, w których jest to ważne, Python jest (niezbyt) tajną bronią.

Poza tym Python jest darmowy. Można w nim pisać wszystko i wszędzie bezpłatnie.

Python działa wszędzie, ponieważ w swojej standardowej bibliotece zawiera mnóstwo przydatnego oprogramowania. W tej książce zaprezentowanych jest wiele przykładów, w których wykorzystana jest standardowa biblioteka oraz liczne użyteczne biblioteki zewnętrzne.

Jednak najważniejszy powód, dla którego warto używać Pythona, jest nietypowy: programiści go lubią i nie uważają stosowania go za zło konieczne do osiągnięcia zamierzonego celu. O tym języku mówi się, że „sam wchodzi do głowy”. Ci, którzy używają innych języków, często twierdzą, że brakuje im funkcjonalności, jakie oferuje Python. Tym różni się ten język od swoich konkurentów.

Kiedy nie używać Pythona?

Python nie jest językiem dobrym na wszystko.

W większości sytuacji kod napisany w Pythonie działa wystarczająco szybko, ale zdarzają się szczególne przypadki, w których jest zbyt wolny. Jeżeli program, który zamierzasz napisać, ma głównie wykonywać obliczenia (używając fachowego żargonu, będzie *ograniczony przez procesor*), lepiej będzie, jeżeli użyjesz języka C, C++, C#, Java, Rust lub Go.

Nie jest to jednak regułą, ponieważ:

- Niektóre algorytmy stosowane w Pythonie są wydajniejsze niż na przykład w C. Ponadto dzięki temu, że programista w Pythonie tworzy kod szybciej niż w innych językach, ma on więcej czasu na eksperymentowanie z alternatywnymi rozwiązaniami.
- W wielu aplikacjach, szczególnie internetowych, kod beczynnie oczekuje, aż serwer wyśle odpowiedź przez sieć. Procesor jest wtedy obciążony w znikomym stopniu i w efekcie różnica pomiędzy szybkością działania programów napisanych w językach statycznym i dynamicznym jest niewielka.
- Interpreter Pythona jest napisany w języku C i można go rozszerzać o ten kod. Ten temat jest poruszony w rozdziale 19.
- Interpreter Pythona jest coraz szybszy. Język Java w swoim dziecięcym wieku był nieznośnie powolny i w jego przyspieszenie włożono mnóstwo wysiłku i pieniędzy. Python nie jest własnością żadnej firmy, więc usprawnienia pojawiają się stopniowo. W rozdziale 19. opisany jest projekt PyPy i uzyskane dzięki niemu efekty.

Czasami jednak wymagania stawiane przed aplikacją są tak wysokie, że za pomocą Pythona nie można ich spełnić, nawet bardzo się starając. Zazwyczaj wyjściem jest wtedy użycie C, C++ lub Javy. Warto również przyjrzeć się językom Go (który jest podobny do Pythona i wydajny jak C) i Rust.

Wersje Pythona 2 i 3

Pewną komplikację stanowi to, że są dwie wersje Pythona. Wersja 2 istnieje od zawsze i jest standardowo instalowana w systemach Linux i macOS. Jest świetna, ale nie idealna. W informatyce, tak jak w wielu innych dziedzinach, niektóre kosmetyczne błędy łatwo jest poprawić, inne trudniej. Wprowadzanie dużych poprawek powoduje, że kolejne wersje przestają być *kompatybilne*. Programy napisane przy użyciu nowszej wersji mogą nie działać w starszych środowiskach i odwrotnie.

Twórca Pythona, Guido van Rossum (<https://www.python.org/~guido>), wraz ze współpracownikami postanowił zebrać wszystkie poprawki i w 2008 r. udostępnił nową wersję języka Python 3. Od tamtej pory Python 2 jest uważany za przeszłość, a Python 3 za przyszłość. Najnowsza podwersja Pythona 2, oznaczona numerem 2.7, będzie jeszcze istnieć przez jakiś czas, ale jest ostatnia. Podwersji 2.8 już nie będzie. Utrzymywanie Pythona 2 i wielu ważnych pakietów zakończyło się w styczniu 2020 r. i od tamtej pory nie są już wprowadzane poprawki (patrz <https://python3statement.org>). Wkrótce z systemów operacyjnych zostanie usunięta wersja 2 i domyślnie będzie instalowana wersja 3. Popularne programy są stopniowo konwertowane do nowej wersji i zdecydowana większość jest już do niej dostosowana. Wszystkie nowe projekty są teraz tworzone w wersji 3.

W niniejszej książce wykorzystana jest wersja 3, która wygląda tu niemal identycznie jak wersja 2. Jedną z najważniejszych różnic polega na tym, że w wersji 3 `print` jest funkcją i jej argumenty trzeba umieszczać w nawiasach. Oprócz tego inaczej są obsługiwane znaki Unicode, o czym będzie mowa w rozdziale 12. Inne różnice są opisywane na bieżąco w tekście.

Instalacja Pythona

Python nie jest domyślnie zainstalowany w każdym systemie. Aby nie robić bałaganu w rozdziale, szczegółowe informacje, jak zainstalować Pythona, zostały umieszczone w dodatku B. Jeżeli używasz innej wersji Pythona niż 3 lub nie wiesz dokładnie jakiej, zajrzyj do tego dodatku i sprawdź, co musisz zrobić. Może to być dość skomplikowane, ale będziesz to musiał zrobić tylko raz.

Uruchomienie Pythona

Po zainstalowaniu środowiska Python 3 możesz go użyć do uruchamiania programów opisanych w tej książce, jak również własnych. Jak się to robi? Poniżej przedstawione są dwa podstawowe sposoby.

- Z małymi programami można łatwo eksperymentować za pomocą wbudowanego **interaktywnego interpretera** (powłoki). Wpisuje się w nim kod wiersz po wierszu i natychmiast uzyskuje wyniki. Dzięki krótkiej drodze między wpisaniem kodu a uzyskaniem wyników można efektywniej eksperymentować. W tej książce używam interpretera do prezentowania funkcjonalności języka. Te same instrukcje możesz wpisywać we własnym środowisku.
- Większe programy zapisuje się w plikach tekstowych, zazwyczaj z rozszerzeniem `.py`, i uruchamia przez wpisanie polecenia `python` i nazwy pliku.

Wypróbujmy teraz obie metody.

Interaktywny interpreter

W większości opisanych w tej książce przykładów wykorzystany jest interaktywny interpreter. Gdy będziesz wpisywał te same instrukcje jak w przykładach i uzyskiwał te same efekty, będzie to oznaczało, że idziesz właściwą drogą.

Interpreter uruchamia się przez wpisanie nazwy głównego pliku wykonywalnego Pythona: `python`, `python3` lub podobnej. W całej książce przyjęte jest założenie, że plik ten nazywa się `python`. Jeżeli w Twoim przypadku nazwa jest inna, wpisz ją wszędzie tam, gdzie w przykładach jest użyta nazwa `python`.

Instrukcje wpisywane w interaktywnym interpreterze i w plikach działają niemal tak samo, z jednym wyjątkiem. Jeżeli w interpreterze wpiszesz instrukcję reprezentującą jakąś wartość, zostanie ona automatycznie wyświetlona. Nie jest to cecha Pythona, tylko interpretera, dzięki której nie trzeba za każdym razem wpisywać funkcji `print()`. Na przykład, gdy w interpreterze wpiszesz liczbę 27, pojawi się ona na ekranie. Jeżeli użyjesz jej w pliku, nic złego się nie stanie, ale liczba ta nigdzie się nie pojawi. Ilustruje to poniższy przykład.

```
$ python
```

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> 27
```

```
27
```



W tym przykładzie symbol \$ oznacza *znak zachęty* do wprowadzenia polecenia (tutaj `python`). Jest on stosowany w przykładach opisanych w tej książce, jednak w Twoim systemie może być inny.

Aby wyświetlić coś w interpreterze, użyj funkcji `print()`, jak niżej:

```
>>> print(27)  
27
```

Jeżeli wypróbowałeś opisane wcześniej przykłady w interaktywnym interpreterze i uzyskałeś takie same wyniki, to znaczy, że uruchomiłeś prawdziwe, choć małe, programy. W kolejnych rozdziałach będziesz stopniowo tworzył coraz dłuższe kody.

Skrypty

Jeżeli liczbę 27 wpiszesz w pliku i uruchomisz go, wtedy na ekranie nic się nie pojawi. Jest to typowe działanie programów, które nie są interaktywne. Aby coś wyświetlić, musisz użyć funkcji `print()`, jak niżej:

```
print(27)
```

Napisz program w Pythonie i uruchom go w następujący sposób:

1. Otwórz edytor tekstowy.
2. Wpisz instrukcję `print(27)`, jak wyżej.

3. Zapisz plik pod nazwą z rozszerzeniem `.py`, na przykład `test.py`. Upewnij się, że zapisałeś go w zwykłym formacie tekstowym, a nie wzbogaconym, na przykład RTF lub Word. Stosowanie rozszerzenia `.py` nie jest obowiązkowe, jednak warto to robić, ponieważ dzięki niemu wiadomo, co to jest za plik.
4. Jeżeli korzystasz z graficznego interfejsu systemu operacyjnego — jak niemal wszyscy — otwórz terminal lub wiersz poleceń².
5. Uruchom program następującym poleceniem:

```
$ python test.py
```

Powinien pojawić się jeden wiersz z wynikiem:

```
27
```

Udało się? Jeżeli tak, to gratuluję Ci napisania i uruchomienia pierwszego programu w języku Python.

Co dalej?

Teraz zaczniesz wpisywać polecenia języka Python zgodnie z obowiązującą w nim składnią. Nie zarzucę Cię od razu wszystkimi regułami, tylko będę je przedstawiał stopniowo w kolejnych rozdziałach.

Podstawowy sposób tworzenia programów w Pythonie polega na wpisywaniu kodu w zwykłym edytorze tekstu lub oknie terminala. W tej książce prezentowany jest zwykły tekst wpisywany w terminalu lub w pliku. Pamiętaj jednak, że dostępnych jest kilka niezłych środowisk IDE (ang. *Integrated Development Environment* — zintegrowane środowisko programistyczne), oferujących wygodę interfejsu graficznego i zaawansowanych technik edytowania i wyświetlania tekstu. Więcej na ten temat dowiesz się w rozdziale 19.

Twoja chwila zen

Każdy język programowania ma własny styl. We wstępie wspomniałem, że istnieje pytoniczny sposób wyrażanie siebie. Interpreter Pythona zawiera w sobie kilkanaście sentencji oddających zwięźle filozofię tego języka (o ile mi wiadomo, Python jest jedynym językiem, który ma coś takiego). Aby przeżyć chwilę zen, wpisz w interaktywnym interpreterze polecenie `import this` i naciśnij klawisz `Enter`. Otrzymasz wynik podobny do poniższego:

```
The Zen of Python, by Tim Peters
```

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.
```

² Jeżeli nie wiesz, co to jest, zajrzyj do dodatku B, w którym opisane są niuanse różnych systemów operacyjnych.

Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Wszystkie przykłady prezentowane w tej książce nawiązują do powyższych zasad.

Już wkrótce

Następny rozdział traktuje o typach danych i zmiennych. Stanowi przygotowanie do dalszych rozdziałów, w których zagłębimy się w szczegóły typów danych i struktury kodu.

Do zrobienia

Niniejszy rozdział jest wprowadzeniem do języka Python — o tym, do czego służy, jak wygląda i jakie jest jego miejsce w informatycznym świecie. Na zakończenie proponuję Ci wykonanie kilku miniprojektów, dzięki którym utrwalisz dotąd zdobytą wiedzę i przygotujesz się do dalszych rozdziałów.

- 1.1. Jeżeli nie zainstalowałeś jeszcze Pythona 3, zrób to teraz. W dodatku B znajdziesz szczegółowe informacje dotyczące Twojego systemu.
- 1.2. Uruchom interaktywny interpreter Pythona 3. Jak poprzednio, szczegółowe informacje są dostępne w dodatku B. Interpreter wyświetli kilka wierszy informacji o sobie i znaki `>>>`, będące zachętą do wpisania instrukcji Pythona.
- 1.3. Poeksperymentuj trochę z interpreterem. Użyj go w charakterze kalkulatora. Wpisz na przykład `8 * 9` i naciśnij *Enter*. Powinien pojawić się wynik 72.
- 1.4. Wpisz liczbę `47` i naciśnij *Enter*. Czy w następnym wierszu pojawiła się ta sama liczba?
- 1.5. Wpisz instrukcję `print(47)` i naciśnij *Enter*. Czy tym razem też pojawiła się ta liczba?

PROGRAM PARTNERSKI

— GRUPY HELION —

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

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA
Helion 

Problemy? Rozwiąż je po pythonowsku!

Python nie jest językiem idealnym, jednak przybywa programistów, którzy uważają go za bliski ideału. Wyróżnia się prostotą i wszechstronnością. Jest wdzięcznym narzędziem do badania danych i tworzenia systemów sztucznej inteligencji, uwielbiają go analitycy, ekonomiści i naukowcy. Może posłużyć do tworzenia stron WWW czy aplikacji specjalnego przeznaczenia. Python należy do najbardziej spójnych i czytelnych języków programowania. Jest przykładem całkiem udanego kompromisu pomiędzy prostotą, łatwością przyswajania i wyjątkową skutecznością. Z pewnością warto się go nauczyć, jednak od początku dobrze jest wpoić sobie nawyki pisania kodu nowoczesnego, wysokiej jakości, zgodnego z dobrą praktyką.

Oto znakomity, przystępny i świetnie napisany podręcznik do nauki Pythona. Opisuje podstawy kodu i struktur danych i stopniowo wprowadza bardziej zaawansowane zagadnienia, takie jak praca z bazami danych i stronami WWW, podstawy działania chmury obliczeniowej, uczenia maszynowego i strumieniowania zdarzeń. Poza standardową biblioteką Pythona przedstawiono tu przydatne zewnętrzne pakiety, dokładniej opisano te najbardziej pomocne. Omówiono dobre praktyki tworzenia, testowania i diagnozowania kodu. Książka zawiera też mnóstwo wskazówek i przykładów kodu. Wyjaśnia pewne szczególne funkcjonalności Pythona, których stosowanie jest o wiele lepszym rozwiązaniem niż adaptowanie technik z innych języków. Nawet jeśli dziś o programowaniu wiesz mniej niż niewiele, dzięki temu podręcznikowi staniesz się prawdziwym pythonowcem!

W książce między innymi:

- podstawy Pythona oraz funkcje, moduły i pakiety
- programowanie zorientowane obiektowo
- praca z bazami danych: relacyjnymi i NoSQL
- klienci internetowe, serwery, interfejsy API i usługi
- zarządzanie programami, procesami i wątkami
- implementacja współbieżności i komunikacji sieciowej

Bill Lubanovic jest ekspertem w dziedzinie technologii informatycznych i programistą. Systemem Unix zajmuje się od 1977 roku, interfejsami graficznymi od 1981 roku, bazami danych od początku lat 90., a siecią WWW od 1993 roku. Mieszka z rodziną w górach Sangre de Sasquatch w Minnesocie w Stanach Zjednoczonych.

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

Sprawdź nasze szkolenia!
SZKOLENIA
AKADEMIA IT & BUSINESS
HELIONSZKOLENIA.PL

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-283-6842-2



INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 89,00 zł