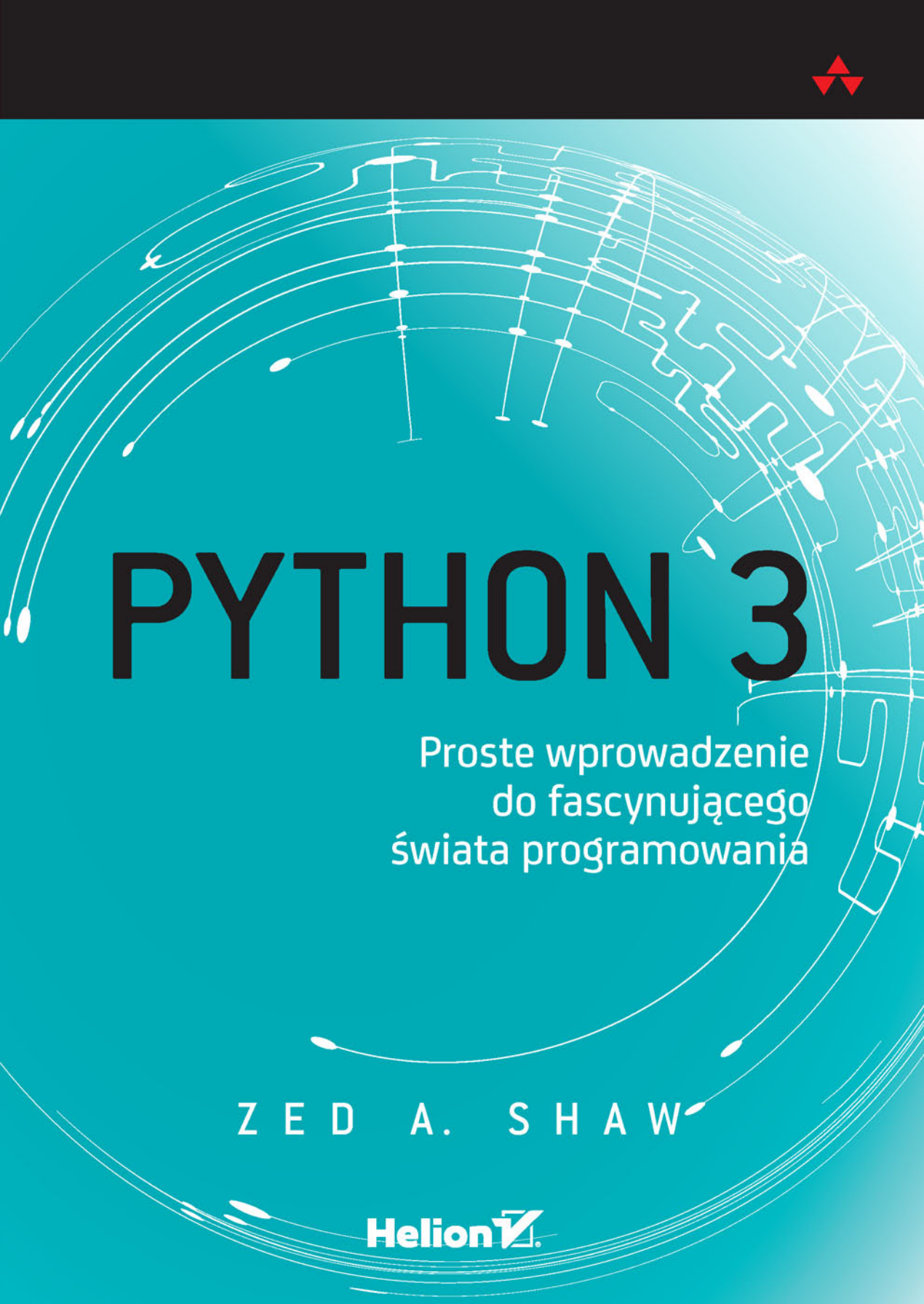

PYTHON 3
Proste wprowadzenie
do fascynującego
świata programowania
ZED A. SHAW
Helion

Tytuł oryginału: Learn Python 3 the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code (Zed Shaw's Hard Way Series)

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-4141-8

Authorized translation from the English language edition, entitled: LEARN PYTHON 3 THE HARD WAY: A VERY SIMPLE INTRODUCTION TO THE TERRIFYINGLY BEAUTIFUL WORLD OF COMPUTERS AND CODE; ISBN 0134692888; by Zed A. Shaw; published by Pearson Education, Inc.

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/pyt3pw
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Przedmowa 15
Ulepszenia w edycji Python 3 15
Trudna droga jest łatwiejsza 16
Czytanie i pisanie 16
Dbałość o szczegóły 16
Dostrzeganie różnic 16
Pytaj, pytaj i pytaj 17
Nie przeklejaj 17
Uwagi na temat ćwiczeń i wytrwałości 17
Podziękowania 18

Ćwiczenie 0. Konfiguracja 20
macOS 20
macOS. Co powinieneś zobaczyć 21
Windows 21
Windows. Co powinieneś zobaczyć 22
Linux 23
Linux. Co powinieneś zobaczyć 23
Szukanie informacji w internecie 24
Ostrzeżenia dla początkujących 24
Alternatywne edytory tekstów 25

Ćwiczenie 1. Dobry pierwszy program 28
Co powinieneś zobaczyć 29
Zrób to sam 31
Typowe pytania 31

Ćwiczenie 2. Komentarze i znaki kratki 34
Co powinieneś zobaczyć 34
Zrób to sam 34
Typowe pytania 35

Ćwiczenie 3. Liczby i działania algebraiczne 36
Co powinieneś zobaczyć 37
Zrób to sam 37
Typowe pytania 37

Ćwiczenie 4. Zmienne i nazwy ... 40
Co powinieneś zobaczyć ... 41
Zrób to sam ... 41
Typowe pytania ... 42

Ćwiczenie 5. Więcej zmiennych i drukowania ... 44
Co powinieneś zobaczyć ... 44
Zrób to sam ... 45
Typowe pytania ... 45

Ćwiczenie 6. Łańcuchy znaków i tekst ... 46
Co powinieneś zobaczyć ... 47
Zrób to sam ... 47
Popsuj kod ... 47
Typowe pytania ... 48

Ćwiczenie 7. Więcej drukowania ... 50
Co powinieneś zobaczyć ... 50
Zrób to sam ... 50
Popsuj kod ... 51
Typowe pytania ... 51

Ćwiczenie 8. Drukowanie, drukowanie ... 52
Co powinieneś zobaczyć ... 52
Zrób to sam ... 53
Typowe pytania ... 53

Ćwiczenie 9. Drukowanie, drukowanie, drukowanie ... 54
Co powinieneś zobaczyć ... 54
Zrób to sam ... 55
Typowe pytania ... 55

Ćwiczenie 10. Co to było? ... 56
Co powinieneś zobaczyć ... 57
Sekwencje ucieczki ... 57
Zrób to sam ... 58
Typowe pytania ... 58

Ćwiczenie 11. Zadawanie pytań ... 60
Co powinieneś zobaczyć ... 60
Zrób to sam ... 61
Typowe pytania ... 61

Ćwiczenie 12. Wyświetlanie podpowiedzi dla użytkownika 62
Co powinieneś zobaczyć 62
Zrób to sam 62
Typowe pytania 63

Ćwiczenie 13. Parametry, rozpakowywanie i zmienne 64
Chwileczkę! Funkcjonalności mają jeszcze inną nazwę 65
Co powinieneś zobaczyć 65
Zrób to sam 66
Typowe pytania 66

Ćwiczenie 14. Znak zachęty i przekazywanie zmiennej 68
Co powinieneś zobaczyć 69
Zrób to sam 69
Typowe pytania 69

Ćwiczenie 15. Czytanie z plików 72
Co powinieneś zobaczyć 73
Zrób to sam 73
Typowe pytania 74

Ćwiczenie 16. Czytanie i zapisywanie plików 76
Co powinieneś zobaczyć 77
Zrób to sam 78
Typowe pytania 78

Ćwiczenie 17. Więcej plików 80
Co powinieneś zobaczyć 80
Zrób to sam 81
Typowe pytania 82

Ćwiczenie 18. Nazwy, zmienne, kod i funkcje 84
Co powinieneś zobaczyć 85
Zrób to sam 86
Typowe pytania 86

Ćwiczenie 19. Funkcje i zmienne 88
Co powinieneś zobaczyć 89
Zrób to sam 89
Typowe pytania 89

Ćwiczenie 20. Funkcje i pliki ... 92
Co powinieneś zobaczyć ... 93
Zrób to sam ... 93
Typowe pytania ... 93

Ćwiczenie 21. Funkcje mogą coś zwracać ... 96
Co powinieneś zobaczyć ... 97
Zrób to sam ... 97
Typowe pytania ... 98

Ćwiczenie 22. Czego nauczyłeś się do tej pory? ... 100
Miej świadomość, czego się uczysz ... 100

Ćwiczenie 23. Łańcuchy znaków, bajty i kodowanie znaków ... 102
Wstępne badanie kodu ... 102
Przełączniki, konwencje i rodzaje kodowania ... 104
Analizujemy dane wyjściowe ... 106
Analizujemy kod ... 106
Bawimy się kodowaniem ... 109
Popsuj kod ... 109

Ćwiczenie 24. Więcej praktyki ... 110
Co powinieneś zobaczyć ... 111
Zrób to sam ... 111
Typowe pytania ... 111

Ćwiczenie 25. Jeszcze więcej praktyki ... 112
Co powinieneś zobaczyć ... 113
Zrób to sam ... 114
Typowe pytania ... 115

Ćwiczenie 26. Gratulacje, rozwiąż test! ... 116
Typowe pytania ... 117

Ćwiczenie 27. Zapamiętywanie logiki ... 118
Wyrażenie logiczne ... 119
Tablice prawdy ... 119
Typowe pytania ... 120

Ćwiczenie 28. Ćwiczymy logikę boolowską ... 122
Co powinieneś zobaczyć ... 124
Zrób to sam ... 124
Typowe pytania ... 124

Ćwiczenie 29. Co, jeśli... 126
Co powinieneś zobaczyć 126
Zrób to sam 127
Typowe pytania 127

Ćwiczenie 30. Else oraz if 128
Co powinieneś zobaczyć 129
Zrób to sam 129
Typowe pytania 129

Ćwiczenie 31. Podejmowanie decyzji 130
Co powinieneś zobaczyć 131
Zrób to sam 131
Typowe pytania 131

Ćwiczenie 32. Pętle i listy 134
Co powinieneś zobaczyć 135
Zrób to sam 136
Typowe pytania 136

Ćwiczenie 33. Pętle while 138
Co powinieneś zobaczyć 139
Zrób to sam 139
Typowe pytania 140

Ćwiczenie 34. Uzyskiwanie dostępu do elementów list 142
Zrób to sam 143

Ćwiczenie 35. Gałęzie i funkcje 144
Co powinieneś zobaczyć 145
Zrób to sam 146
Typowe pytania 146

Ćwiczenie 36. Projektowanie i debugowanie 148
Zasady dotyczące instrukcji if 148
Zasady dotyczące pętli 148
Wskazówki dotyczące debugowania 149
Praca domowa 149

Ćwiczenie 37. Przegląd symboli 150
Słowa kluczowe 150
Typy danych 151

Sekwencje ucieczki ... 152
Formatowanie łańcuchów znaków w starym stylu ... 152
Operatory ... 153
Czytanie kodu ... 154
Zrób to sam ... 155
Typowe pytania ... 155

Ćwiczenie 38. Operacje na listach ... 156
Co powinieneś zobaczyć ... 157
Co mogą robić listy ... 158
Kiedy używać list ... 159
Zrób to sam ... 159
Typowe pytania ... 160

Ćwiczenie 39. Słowniki, piękne słowniki ... 162
Przykład słownika ... 163
Co powinieneś zobaczyć ... 164
Co mogą robić słowniki ... 165
Zrób to sam ... 166
Typowe pytania ... 166

Ćwiczenie 40. Moduły, klasy i obiekty ... 168
Moduły są jak słowniki ... 168
Klasy są jak moduły ... 169
Obiekty są jak import ... 170
Różne sposoby pobierania elementów ... 171
Pierwszy przykład klasy ... 172
Co powinieneś zobaczyć ... 172
Zrób to sam ... 172
Typowe pytania ... 173

Ćwiczenie 41. Uczymy się mówić obiektowo ... 174
Ćwiczymy słówka ... 174
Ćwiczymy zdania ... 174
Ćwiczenie łączone ... 175
Test z czytania ... 175
Tłumaczenie ze zdań na kod ... 177
Poczytaj jeszcze więcej kodu ... 178
Typowe pytania ... 178

Ćwiczenie 42. Jest, ma, obiekty i klasy 180
Jak to wygląda w kodzie 181
Na temat class Nazwa(object) 183
Zrób to sam 183
Typowe pytania 184

Ćwiczenie 43. Podstawowa analiza obiektowa i projekt 186
Analiza prostego silnika gry 187
Opisz lub rozrysuj problem 187
Wyodrębnij kluczowe pojęcia i zbadaj je 188
Utwórz hierarchię klas i mapę obiektów dla koncepcji 188
Zakoduj klasy i napisz test, aby je uruchomić 189
Powtórz i udoskonal 191
Z góry do dołu i z dołu do góry 191
Kod gry Goci z planety Percal 25 192
Co powinieneś zobaczyć 198
Zrób to sam 198
Typowe pytania 199

Ćwiczenie 44. Porównanie dziedziczenia i kompozycji 200
Co to jest dziedziczenie 200
Dziedziczenie domyślne 201
Bezpośrednie nadpisanie 202
Zmiana zachowania przed lub po 202
Połączenie wszystkich trzech sposobów 203
Dlaczego super() 205
Używanie super() z __init__ 205
Kompozycja 205
Kiedy używać dziedziczenia, a kiedy kompozycji 207
Zrób to sam 207
Typowe pytania 207

Ćwiczenie 45. Tworzysz grę 210
Ocenianie napisanej gry 210
Styl funkcji 211
Styl klas 211
Styl kodu 212
Dobre komentarze 212
Oceń swoją grę 213

Ćwiczenie 46. Szkielet projektu ... 214
Konfiguracja w systemach macOS i Linux ... 214
Konfiguracja w systemie Windows 10 ... 215
Tworzenie szkieletu katalogu projektów ... 217
Ostateczna struktura katalogów ... 218
Testowanie konfiguracji ... 219
Używanie szkieletu ... 220
Wymagany quiz ... 220
Typowe pytania ... 220

Ćwiczenie 47. Zautomatyzowane testowanie ... 222
Pisanie przypadku testowego ... 222
Wytyczne testowania ... 224
Co powinieneś zobaczyć ... 224
Zrób to sam ... 225
Typowe pytania ... 225

Ćwiczenie 48. Zaawansowane wprowadzanie danych przez użytkownika ... 226
Nasz leksykon gry ... 226
Rozkładanie zdań na części ... 227
Krotki leksykonu ... 227
Skanowanie danych wejściowych ... 227
Wyjątki i liczby ... 227
Wyzwanie „najpierw przygotuj testy” ... 228
Co powinieneś przetestować ... 229
Zrób to sam ... 231
Typowe pytania ... 231

Ćwiczenie 49. Tworzenie zdań ... 232
Dopasowywanie i podglądanie ... 232
Gramatyka zdania ... 233
Słowo o wyjątkach ... 233
Kod parsera ... 233
Zabawa z parserem ... 236
Co powinieneś przetestować ... 237
Zrób to sam ... 237
Typowe pytania ... 237

Ćwiczenie 50. Twoja pierwsza strona internetowa 238
Instalowanie frameworku flask 238
Tworzenie prostego projektu „Witaj, świecie” 238
Co się tutaj dzieje 240
Naprawianie błędów 240
Tworzenie podstawowych szablonów 241
Zrób to sam 243
Typowe pytania 243

Ćwiczenie 51. Pobieranie danych wejściowych z przeglądarki 246
Jak działa sieć 246
Jak działają formularze 248
Tworzenie formularzy HTML 249
Tworzenie szablonu układu 251
Pisanie zautomatyzowanych testów dla formularzy 252
Zrób to sam 254
Popsuj kod 254

Ćwiczenie 52. Początek Twojej gry internetowej 256
Refaktoryzacja gry z ćwiczenia 43. 256
Tworzenie silnika 261
Twój egzamin końcowy 263
Typowe pytania 264

Następne kroki 266
Jak uczyć się dowolnego języka programowania 267

Porada starego programisty 270

Dodatek. Przyspieszony kurs wiersza poleceń 272
Wprowadzenie: zamknij się i zacznij używać powłoki 272
Jak korzystać z tego dodatku 273
Będziesz musiał zapamiętywać rzeczy 273
Konfiguracja 274
Zadanie 274
Czego się nauczyłeś 275
Zadanie dodatkowe 275
Ścieżki, foldery, katalogi (pwd) 277
Zadanie 277
Czego się nauczyłeś 278
Zadanie dodatkowe 278

Jeśli się zgubisz ... 279
Zadanie ... 279
Czego się nauczyłeś ... 279
Tworzenie katalogu (mkdir) ... 279
Zadanie ... 279
Czego się nauczyłeś ... 280
Zadanie dodatkowe ... 281
Zmienianie katalogu (cd) ... 281
Zadanie ... 281
Czego się nauczyłeś ... 284
Zadanie dodatkowe ... 284
Listowanie katalogu (ls) ... 285
Zadanie ... 285
Czego się nauczyłeś ... 287
Zadanie dodatkowe ... 288
Usuwanie katalogu (rmdir) ... 288
Zadanie ... 288
Czego się nauczyłeś ... 290
Zadanie dodatkowe ... 290
Poruszanie się po katalogach (pushd, popd) ... 290
Zadanie ... 291
Czego się nauczyłeś ... 292
Zadanie dodatkowe ... 293
Tworzenie pustych plików (touch/New-Item) ... 293
Zadanie ... 293
Czego się nauczyłeś ... 294
Zadanie dodatkowe ... 294
Kopiowanie pliku (cp) ... 294
Zadanie ... 294
Czego się nauczyłeś ... 296
Zadanie dodatkowe ... 297
Przenoszenie pliku (mv) ... 297
Zadanie ... 297
Czego się nauczyłeś ... 298
Zadanie dodatkowe ... 298
Przeglądanie pliku (less/more) ... 299
Zadanie ... 299
Czego się nauczyłeś ... 300
Zadanie dodatkowe ... 300

Strumieniowanie pliku (cat) 300
Zadanie 300
Czego się nauczyłeś 301
Zadanie dodatkowe 301
Usuwanie pliku (rm) 301
Zadanie 301
Czego się nauczyłeś 303
Zadanie dodatkowe 303
Wyjście z terminala (exit) 303
Zadanie 303
Czego się nauczyłeś 303
Zadanie dodatkowe 303
Następne kroki z wierszem poleceń 304
Zasoby dla uniksowej powłoki bash 304

Skorowidz 305

ĆWICZENIE 1

Dobry pierwszy program

OSTRZEŻENIE! Jeśli pominąłeś ćwiczenie 0, nie korzystasz z tej książki we właściwy sposób. Próbujesz używać IDLE lub IDE? W ćwiczeniu 0 zaznaczyłem, żebyś nie używał tych środowisk, więc nie powinieneś tego robić. Zatem proszę, wróć do tego ćwiczenia i przeczytaj je.

Powinieneś poświęcić sporo czasu na ćwiczenie 0, żeby nauczyć się, jak zainstalować edytor tekstu, uruchomić go, uruchomić terminal i pracować z oboma narzędziami. Jeśli tego nie zrobiłeś, nie kontynuuj bieżącego ćwiczenia, ponieważ nie spędzisz miło czasu. To jedyny raz, kiedy rozpoczynam ćwiczenie ostrzeżeniem, że nie powinieneś niczego pomijać lub wybiegać do przodu.

Wpisz poniższy tekst w pojedynczym pliku o nazwie *ex1.py*. Python działa najlepiej z plikami z rozszerzeniem *.py*.

ex1.py

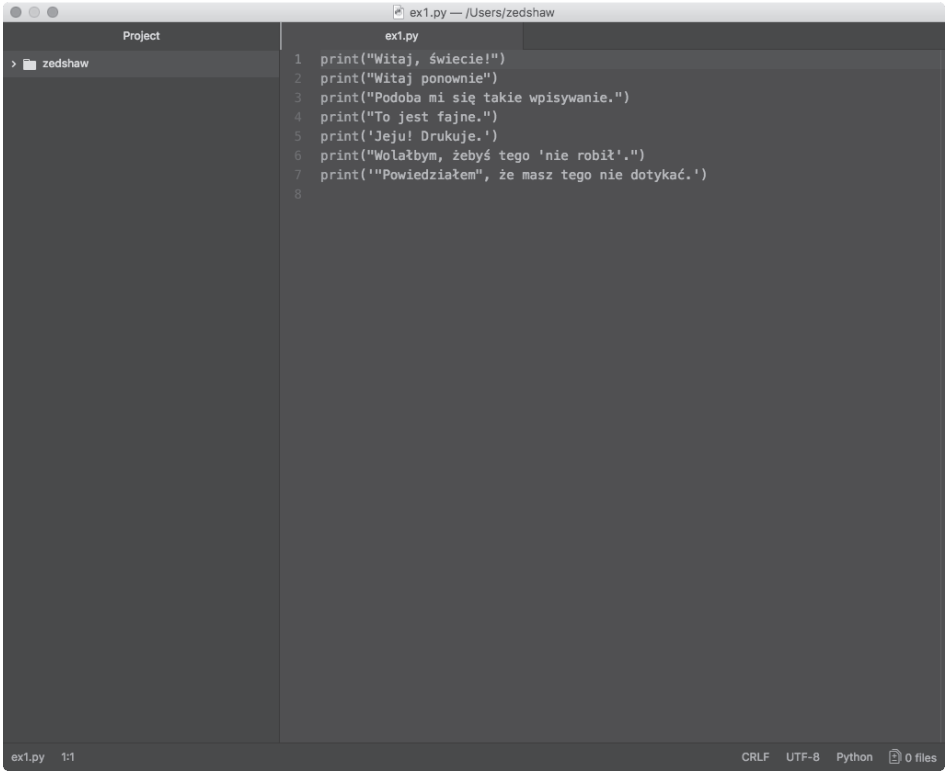
```
1  print("Witaj, świecie!")
2  print("Witaj ponownie")
3  print("Podoba mi się takie wpisywanie.")
4  print("To jest fajne.")
5  print('Jeju! Drukuje.')
6  print("Wolałbym, żebyś tego 'nie robił'.")
7  print('"Powiedziałem", że masz tego nie dotykać.')
```

Okno edytora tekstu Atom powinno wyglądać podobnie na wszystkich platformach (zobacz rysunek na następnej stronie).

Nie przejmuj się, jeśli Twój edytor nie wygląda dokładnie tak samo — powinien wyglądać dość podobnie. Możesz mieć trochę inny nagłówek okna, może nieco inne kolory, a w lewym panelu nie będzie wyświetlać się nazwa projektu *zedshaw*, ale zamiast tego wyświetli się katalog, którego użyłeś do zapisywania plików. Wszystkie te różnice są w porządku.

Podczas tworzenia tego pliku powinieneś pamiętać o następujących kwestiach.

1. Nie wpisywałem numerów linii po lewej stronie. Stosuję je w listingach, żebym mógł odwoływać się do konkretnych linii, na przykład: „Zobacz linię 5.”. W skryptach Pythona nie wpisuje się numerów linii.
2. Na początku każdej linii znajduje się polecenie `print`, które wygląda dokładnie tak samo jak w moim pliku *ex1.py*. „Dokładnie” oznacza dokładnie, a nie mniej więcej tak samo. Aby kod zadziałał, musi się zgadzać każdy pojedynczy znak. Kolor nie ma znaczenia, tylko wpisywane znaki.



W terminalu w systemie macOS lub Linux uruchom ten plik, wpisując następujące polecenie:

```
python3.6 ex1.py
```

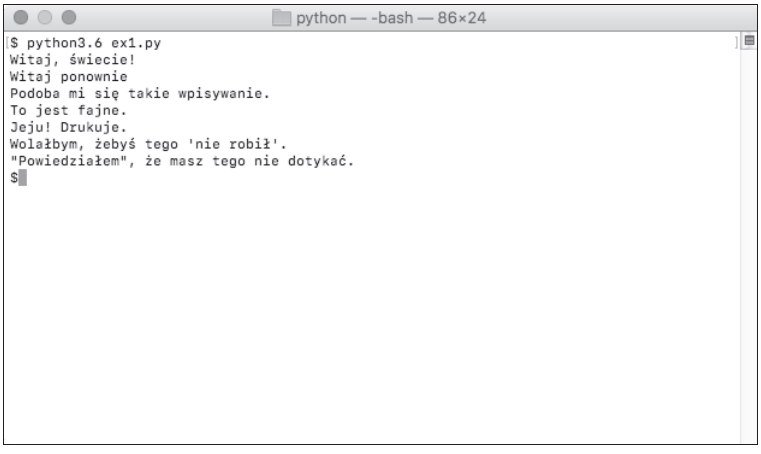
Pamiętaj, że w systemie Windows *zawsze* wpisujesz `python` zamiast `python3.6`, tak jak poniżej:

```
python ex1.py
```

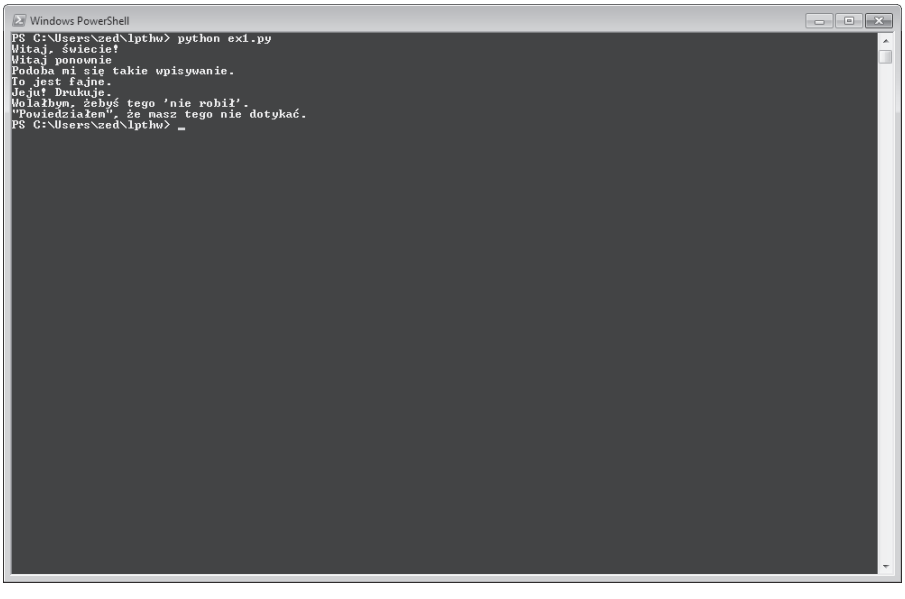
Jeśli zrobiłeś wszystko prawidłowo, powinieneś zobaczyć to samo, co przedstawiono w podrozdziale „Co powinieneś zobaczyć" tego ćwiczenia. Jeśli tego nie widzisz, zrobiłeś coś źle. Nie, komputer się nie pomylił.

Co powinieneś zobaczyć

Oto, co powinieneś zobaczyć w terminalu w systemie macOS.



To powinieneś zobaczyć w PowerShell w systemie Windows.



Przed poleceniem `python3.6 ex1.py` mogą być wyświetlane różne nazwy, ale najważniejsze jest to, żebyś po wpisaniu tego polecenia zobaczył dane wyjściowe takie same jak moje.

Jeśli wystąpi błąd, będzie to wyglądać następująco.

```
$ python3.6 python/ex1.py
  File "python/ex1.py", line 3
    print("Podoba mi się takie wpisywanie.
                                           ^
SyntaxError: EOL while scanning string literal
```

Ważne jest, abyś potrafił odczytywać komunikaty o błędach, ponieważ będziesz popełniał wiele takich błędów. Nawet ja popełniam ich sporo. Przyjrzyjmy się temu linia po linii.

1. Wpisaliśmy nasze polecenie w terminalu, aby uruchomić skrypt *ex1.py*.
2. Python poinformował nas, że plik *ex1.py* ma błąd w linii 3.
3. Wyświetlił tę linię kodu, abyśmy mogli ją zobaczyć.
4. Następnie umieścił znak wstawiania (^), aby wskazać, gdzie jest problem. Zwróciłeś uwagę na brakujący znak podwójnego cudzysłowu (")?
5. Na koniec wyświetlił komunikat `SyntaxError` i podpowiedział, co może być błędem. Zazwyczaj te komunikaty są bardzo tajemnicze, ale jeśli skopiujesz dany tekst do wyszukiwarki, znajdziesz kogoś innego, kto napotkał ten sam błąd, i prawdopodobnie dowiesz się, jak go naprawić.

Zrób to sam

Podrozdziały zatytułowane „Zrób to sam" zawierają zadania, które powinieneś *spróbować* wykonać. Jeśli na razie nie jesteś w stanie poradzić sobie z danym ćwiczeniem, pomiń je i wróć do niego później.

W tym ćwiczeniu spróbuj zrobić trzy rzeczy.

1. Spraw, aby Twój skrypt wydrukował jeszcze jedną linię.
2. Spraw, aby Twój skrypt wydrukował tylko jedną z linii.
3. Umieść znak kratki (#) na początku linii. Co on powoduje? Spróbuj się dowiedzieć, do czego służy ten znak.

Odtąd nie będę już wyjaśniał, na czym polega każde ćwiczenie, chyba że dane ćwiczenie będzie się różnić od pozostałych.

OSTRZEŻENIE! Znak kratki jest nazywany również „krzyżykiem", „hashem", „płotkiem" i tak dalej. Wybierz tę nazwę, która Cię relaksuje.

Typowe pytania

Są to *faktyczne* pytania, które prawdziwi studenci zadali podczas wykonywania tego ćwiczenia.

Czy mogę używać IDLE? Nie. Powinieneś używać terminala w systemach macOS i Linux oraz programu PowerShell w systemie Windows, tak jak robię w tym ćwiczeniu. Jeśli nie wiesz, jak korzystać z tych narzędzi, możesz przeczytać dodatek „Przyspieszony kurs wiersza poleceń".

Jak uzyskać kolory w edytorze? Najpierw zapisz plik jako plik z rozszerzeniem *.py*, na przykład *ex1.py*. Wtedy podczas wpisywania pojawią się kolory.

Gdy próbuję uruchomić plik *ex1.py*, otrzymuję komunikat `SyntaxError: invalid syntax`. Prawdopodobnie uruchomiłeś Pythona, a następnie próbujesz ponownie wpisać polecenie `python` w celu uruchomienia pliku. Zamknij terminal, uruchom go ponownie i od razu wpisz jedynie `python3.6 ex1.py`.

Otrzymuję komunikat `can't open file 'ex1.py': [Errno 2] No such file or directory`. Musisz znajdować się w tym samym katalogu, co utworzony plik. Aby przejść do właściwego katalogu, użyj najpierw polecenia `cd`. Jeśli zapisałeś plik na przykład w lokalizacji */lpthw/ex1.py*, przejdź do niej za pomocą polecenia *cd /lpthw/* i dopiero wtedy wpisz polecenie `python3.6 ex1.py`. Jeśli nie wiesz, co to wszystko oznacza, zapoznaj się z dodatkiem „Przyspieszony kurs wiersza poleceń".

Mój plik nie działa. Po prostu otrzymuję z powrotem nagłówek wiersza poleceń bez żadnych danych wyjściowych. Najprawdopodobniej potraktowałeś zbyt dosłownie kod z mojego pliku *ex1.py* i uznałeś, że `print("Witaj, świecie!")` oznacza, że w pliku należy wpisać tylko `"Witaj, świecie!"` bez słowa `print`. Twój plik musi być *dokładnie* taki sam jak mój.

ĆWICZENIE 2

Komentarze i znaki kratki

Komentarze w Twoich programach są bardzo ważne. Informują, co robi dany fragment kodu, i są używane do wyłączania części programu, jeśli trzeba je tymczasowo usunąć.

Komentarzy w Pythonie używa się w poniższy sposób.

ex2.py

```
1  # Komentarz; ułatwia późniejsze czytanie programu.
2  # Wszystko, co znajduje się po znaku #, jest przez Pythona ignorowane.
3
4  print("Mógłbym napisać taki kod.") # a komentarz po kodzie będzie ignorowany
5
6  # Komentarza możesz również użyć do "wyłączenia", czyli wykomentowania kodu:
7  # print("To nie zostanie uruchomione.")
8
9  print("To zostanie uruchomione.")
```

Od tej pory będę pisać kod właśnie w taki sposób. Powinieneś zrozumieć, że nie wszystko musi być dosłowne. Twój ekran i program mogą wyglądać inaczej, ale najważniejszy jest umieszczany w pliku tekst, który piszesz w edytorze. W rzeczywistości mógłbym pracować z dowolnym edytorem tekstu i rezultat byłyby taki sam.

Co powinieneś zobaczyć

Ćwiczenie 2. — sesja

```
$ python3.6 ex2.py
Mógłbym napisać taki kod.
To zostanie uruchomione.
```

Tym razem również nie pokażę Ci zrzutów ekranu wszystkich możliwych terminali. Powinieneś zrozumieć, że powyższy listing nie jest dosłownym przełożeniem tego, jak będą wyglądać Twoje dane wyjściowe. Musisz natomiast koncentrować się na tekście wyświetlanym między pierwszą linią `$ python3.6...` i ostatnią linią ze znakiem `$`.

Zrób to sam

1. Sprawdź, czy dobrze rozumiesz przeznaczenie znaku `#` i upewnij się, że znasz różne jego nazwy (na przykład płotek czy hash).
2. Przeanalizuj każdą linię kodu z pliku *ex2.py* wstecz. Zacznij od ostatniej linii i sprawdzaj każde słowo, które powinieneś był wpisać.

3. Czy znalazłeś więcej błędów? Usuń je.
4. Przeczytaj głośno to, co wpisałeś powyżej, wymawiając przy tym pełną nazwę każdego znaku specjalnego. Znalazłeś więcej błędów? Usuń je.

Typowe pytania

Czy na pewno znak `#` nazywany jest również hashem? Nazywam ten znak kratką, ponieważ jest to nazwa rozpoznawalna w każdym kraju. Każdy kraj uważa swoją nazwę dla tego znaku za najważniejszą i najtrafniejszą. Dla mnie jest to po prostu arogancja. Trzeba wyluzować i skoncentrować się na ważniejszych sprawach, takich jak nauka kodowania.

Dlaczego znak `#` w poleceniu `print("Hej # tam.")` nie jest ignorowany? Znak `#` w tym kodzie znajduje się wewnątrz łańcucha znaków i dlatego będzie stanowił część tego łańcucha, który kończy się na znaku `"`. Znaki kratki w łańcuchach są uznawane za zwykłe znaki, a nie komentarze.

Jak wykomentować wiele linii? Umieść znak `#` przed każdą z nich.

Nie wiem, jak wpisać znak `#` na mojej klawiaturze, która ma inny układ językowy, charakterystyczny dla mojego kraju. Jak mogę to zrobić? W niektórych krajach do wpisywania znaków obcych dla danego języka używa się kombinacji przycisku *Alt* lub *Shift* z innymi przyciskami klawiatury. Informacji o sposobie wpisywania znaku `#` na Twojej klawiaturze będziesz musiał poszukać w internecie.

Dlaczego muszę czytać kod wstecz? Dzięki tej sztuczce Twój mózg nie będzie przywiązywał się do znaczenia poszczególnych części kodu i będziesz mógł przetworzyć każdy fragment bardzo dokładnie. Jest to przydatna technika sprawdzania błędów.

ĆWICZENIE 3

Liczby i działania algebraiczne

Każdy język programowania ma własny sposób radzenia sobie z liczbami i działaniami algebraicznymi. Nie martw się: programiści często kłamią, że są geniuszami matematycznymi, podczas gdy naprawdę nie są. Gdyby nimi byli, zajmowaliby się matematyką, a nie pisaniem dziwacznych frameworków webowych, aby zarobić na sportowe auta.

To ćwiczenie zawiera wiele symboli matematycznych. Nazwijmy je od razu, żebyś wiedział, o czym mowa. Podczas wpisywania poszczególnych symboli wymawiaj ich pełne nazwy. Kiedy Ci się to znudzi, możesz przestać. Oto nazwy symboli:

- : plus,
- -: minus,
- /: ukośnik,
- *: gwiazdka,
- %: procent,
- <: mniejsze niż,
- >: większe niż,
- <=: mniejsze lub równe,
- >=: większe lub równe.

Zwróciłeś uwagę, że brakuje operacji arytmetycznych? Po wpisaniu kodu tego ćwiczenia wróć do powyższego zestawienia, zastanów się, jakie są funkcje poszczególnych symboli, i uzupełnij tabelę, na przykład znak + to dodawanie.

ex3.py

```
print("Policzę teraz pogłowie mojego drobiu:")

print("Kury", 25 + 30 / 6)
print("Koguty", 100 - 25 * 3 % 4)

print("Teraz policzę jaja:")

print(3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6)

print("Czy to prawda, że 3 + 2 < 5 - 7?")

print(3 + 2 < 5 - 7)

print("Ile to jest 3 + 2?", 3 + 2)
print("Ile to jest 5 - 7?", 5 - 7)

print("Oj, to dlatego to daje False.")

```

```
19    print("Poćwiczmy jeszcze trochę.")
20
21    print("Czy to jest większe?", 5 > -2)
22    print("Czy to jest większe lub równe?", 5 >= -2)
23    print("Czy to jest mniejsze lub równe?", 5 <= -2)
```

Zanim uruchomisz plik, upewnij się, że wpisałeś dokładnie to, co wypisałem w listingu. Porównaj każdą linię swojego pliku z moim.

Co powinieneś zobaczyć

Ćwiczenie 3. — sesja

```
$ python3.6 ex3.py
Policzę teraz pogłowie mojego drobiu:
Kury 30.0
Koguty 97
Teraz policzę jaja:
6.75
Czy to prawda, że 3 + 2 < 5 - 7?
False
Ile to jest 3 + 2? 5
Ile to jest 5 - 7? -2
Oj, to dlatego to daje False.
Poćwiczmy jeszcze trochę.
Czy to jest większe? True
Czy to jest większe lub równe? True
Czy to jest mniejsze lub równe? False
```

Zrób to sam

1. Nad każdą linią kodu użyj znaku `#`, aby napisać komentarz wyjaśniający, co robi.
2. Pamiętasz z ćwiczenia 0, jak uruchamiałeś `python3.6`? Ponownie uruchom polecenie `python3.6` w ten sam sposób i za pomocą operatorów matematycznych użyj Pythona jako kalkulatora.
3. Znajdź coś, co musisz obliczyć, i napisz nowy plik *.py*, który to robi.
4. Napisz ćwiczenie *ex3.py* od nowa, używając liczb zmiennoprzecinkowych, aby obliczenia były bardziej dokładne. Liczbą zmiennoprzecinkową jest na przykład *20.0*.

Typowe pytania

Dlaczego znak `%` to operacja modulo, a nie procent? Głównie dlatego, że takie przeznaczenie dla tego symbolu wybrali projektanci. W normalnym zapisie matematycznym prawidłowo odczytalibyśmy to oczywiście jako procent. Jednak w programowaniu obliczenia procentowe są zazwyczaj wykonywane przy użyciu prostego dzielenia i operatora `/`. Modulo jest inną operacją, która po prostu używa symbolu `%`.

Jak działa modulo (%)? Inaczej powiedzielibyśmy, że „*X* podzielone przez *Y* daje resztę *J*", na przykład „100 podzielone przez 16 daje resztę 4". Wynik operacji modulo to wartość *J*, czyli reszta z dzielenia.

Jaka jest kolejność wykonywania działań? W języku angielskim do określenia kolejności wykonywania działań używa się akronimu **PEMDAS** (ang. *Parentheses Exponents Multiplication Division Addition Subtraction*), co oznacza: nawiasy, potęgowanie, mnożenie, dzielenie, dodawanie, odejmowanie. Ta kolejność jest również przestrzegana w Pythonie. Częstym błędem jest potraktowanie wskazówki PEMDAS jako ścisłej kolejności, czyli „Najpierw *P* (nawiasy), potem *E* (potęgowanie), potem *M* (mnożenie), potem *D* (dzielenie), potem *A* (dodawanie) i na koniec *S* (odejmowanie)". Właściwą kolejnością jest wykonywanie mnożenia i dzielenia (M&D) w jednym kroku, od lewej do prawej, a następnie dodawania i odejmowania (A&S) w jednym kroku, od lewej do prawej. Możemy więc zapisać PEMDAS jako PE(M&D)(A&S).

Skorowidz

A

adres URL, 247, 249
  przekazywanie parametrów, 249
algebra Boole'a, 120
alternatywa, 119, 151
argument wiersza poleceń, 64, 65, 66
ASCII, 104, 105
Atom, 20, 23, 25
atrybut, 174

B

bajt, 104
biblioteka, 65
bit, 104
blok
  except, 228
  try, 228
  try-except, 231
błąd, 30
  EOL while scanning string literal, 82
  importu, 231
  invalid syntax, 69
  komunikat, 31
  not enough values to unpack,, 66
  not enough values to unpack., 70
  parsowania, 233

C

CLI, 284
command line interface, *Patrz:* CLI

D

dane
  struktura, *Patrz:* struktura danych
  wprowadzanie, 226
DBES, 107
debugger, 149
dictionary, *Patrz:* słownik
działanie kolejność wykonywania, 38
dziecko, *Patrz:* klasa potomna
dziedziczenie, 174, 200, 203, 205, 207
  domyślne, 201
  wielokrotne, 200, 205, 207
dzwonek, 152

E

edytor tekstu, 25, 76
  Atom, *Patrz:* Atom
  Emacs, 25
  Vim, 25
escape sequence, *Patrz:* sekwencja ucieczki

F

format string, *Patrz:* łańcuch znaków sformatowany
formularz, 246, 248, 249
framework
  flask, 238, 241, 253
    dziennik, 239
    instalowanie, 238
    tryb debuggera, 241
  nose, 215, 219
  testowy, 215
  webowy, 238
f-string, 46, 152
funkcja, 84, 186
  __init__, 205, 211
  anonimowa, 151
  assert_equal, 224
  assert_raises, 237
  die, 148
  format, 46, 52
  input, 61, 62, 68
  int, 228
  join, 160
  len, 82
  liczba argumentów, 90
  lista kontrolna, 86
  nadpisanie, 202
  nazwa, 86, 211
  range, 132, 136, 160
  readline, 94
  round, 45

funkcja
  super, 202, 203, 205
  tworzenie, 84, 211
  wartość, 96
  wywoływanie, 89

H

HTML, 242, 249

I

IDE, 25
IDLE, 26
importowanie, 64
instancja, 170, 174, 180
instrukcja
  elif, 129, 130, 131, 150
  else, 129, 130, 150
  if, 126, 127, 128, 130, 131, 151
  if-else, 131
Integrated Development Environment, *Patrz:* IDE
interfejs API, 253

J

język, 105
  programowania, 270
    JavaScript, 208, 267
    nauka, 267
    obiektowego, 168
    prototypowy, 208
    Ruby, 136, 267

K

katalog
  listowanie, 285
  tworzenie, 279
  usuwanie, 288
  zmiana, 281
klasa, 150, 168, 169, 171, 174, 186, 208
  bazowa, 192, 201
  hierarchia, 186, 188
  nadrzędna, 200
  nazwa, 211
  potomna, 201
  tworzenie, 189
kodek, 104
kodowanie, 104
kolejność
  rozwiązywania metod, *Patrz:* MRO
  wykonywania działań, 38
komentarz, 34, 212
  ## ??, 184
  dokumentujący, 114
kompozycja, 174, 207
koniunkcja, 119, 150
krotka, 227, 232

L

liczba
  całkowita, 152
  dziesiętna, 152
  jako łańcuch znaków, 61
  kardynalna, 142, 159
  konwertowanie, 227
  ósemkowa, 152
  szesnastkowa, 152
  zmiennoprzecinkowa, 37, 41, 152
    zaokrąglenie, 45
lista, 134, 136, 156, 157, 158, 166
  dwuwymiarowa, 136
  element, 142
  kopiowanie, 178
  tworzenie, 134
  wycinek, 178
  zastosowania, 159
logika, 118
  boolowska, 122

Ł

łańcuch znaków, 35, 44, 46, 102, 107, 157
  formatowanie, 46, 52, 152
  kod formatowania, 152
  literał sformatowany, 46
  sformatowany, 44
  z osadzonymi zmiennymi, 44

M

method resolution order, *Patrz:* MRO
metoda, 211
  post, 253

moduł, 65, 168, 169, 207
  doctest, 225
MRO, 205

N

negacja, 119, 151
notacja camelCase, 211

O

obiekt, 170, 174, 181, 186, 208
  pliku, 74
operacja
  append, 136
  modulo, 37
operator, 119, 153
  inkrementacji, 127
  logiczny, 119
oprogramowania konfiguracja, 214, 215

P

Parentheses Exponents Multiplication Division Addition Subtraction, *Patrz:* łańcuch znaków
parser, 233, 236
PEMDAS, 38
pętla
  for, 134, 140, 148, 150, 151
  while, 138, 140, 148, 151, 160, 191
plik, 93
  .py, 64
  HTML, 242, 249
  kasowanie, 301
  konkatenacja, 81
  kopiowanie, 80, 296
  languages.txt, 102
  modyfikator trybu, 78
  odczyt, 76
  opróżnianie, 76
  otwieranie, 72, 74
  przenoszenie, 297
  pusty, 293
  tekstowy, 72, 298
  tryb, 78
  zamykanie, 76
  zapisywanie, 76
podpowiedź, 62
polecenie
  apropos, 276
  cat, 81, 276, 300
  cd, 275, 276, 279, 281
  close, 76
  cp, 276, 294
  dir, 276
  echo, 81, 276, 277
  env, 276
  exit, 146, 276, 277, 303
  export, 276
  find, 276
  forfiles, 276
  format, 52
  grep, 276
  help, 277
  helpctr, 277
  hostname, 275, 276
  import, 80
  less, 276, 298, 299
  ls, 275, 276, 285, 287
  man, 276
  mkdir, 275, 276, 279, 280, 293
  more, 276, 298
  mv, 276, 297
  New-Item, 293
  nosetests, 223, 224, 225
  open, 72
  pip, 214, 215
  pip3.6, 214
  popd, 276, 290, 292
  print, 28, 115, 149
  pushd, 276, 290, 292
  pwd, 275, 276, 277, 278, 279
  pydoc, 63
  pydoc input, 62
  python, 219
  quit, 114
  read, 76
  readline, 76
  return, 115
  rm, 301
  rmdir, 275, 276, 288, 289
  robocopy, 276
  runas, 277
  seek, 76, 93, 94
  select-string, 277
  set, 277
  setuptools, 214
  sudo, 276

polecenie
touch, 293
truncate, 76, 78
type, 276
virtualenv, 214, 215
write, 76
xargs, 276
PowerShell, 20, 21, 30, 274, 304
powłoka, 274
bash, 20, 304
programowanie, 270
obiektowe, 159, 168, 186, 200
z dołu do góry, 191
z góry na dół, 186, 191
projekt szkielet, 214, 217, 218, 220
protokół, 247
przeglądarka, 239, 240, 247

R

refaktoryzacja, 256
rodzic, *Patrz:* klasa nadrzędna

S

sekwencja ucieczki, 56, 57, 152
serwer, 248, 254
składnia .format, 52
skrypt, 64
słownik, 150, 162, 165, 166, 168, 169
mapowanie, 163, 165
słowo kluczowe
and, 150
as, 150
assert, 150
break, 150
class, 150, 174
continue, 150
def, 84, 150
del, 150, 163
elif, 150
else, 150
except, 150, 151, 228
exec, 150
False, 53, 119
finally, 150
for, 150
from, 150
global, 150
if, 151
import, 151
in, 151
is, 151
lambda, 151
not, 151
or, 151
pass, 151
print, 151
raise, 151, 233
return, 96, 151
True, 53, 119
try, 151, 228
while, 151
with, 151
yield, 151
standard
ASCII, *Patrz:* ASCII
Unicode, 105, 106
string, *Patrz:* łańcuch znaków
struktura danych, 158, 165
collections.OrderedDict, 166
system
Big5, 109
UTF-8, 102, 106
szkielet, 214, 217, 218, 220

Ś

ścieżka, 281
środowisko programistyczne
virtualenv, 214
zintegrowane, *Patrz:* IDE

T

tablica, 136
prawdy, 119, 122
tabulator, 152
terminal, 20, 23, 29, 274
test
dla formularza, 252
fałszywa sesja, 264
jednostkowy, 222, 227
uruchomienie, 223, 224, 225
zautomatyzowany, 222
token błędu, 227
typ
bytes, 102, 151
danych, 151

dicts, 151
False, 151
floats, 151
lists, 151
None, 151
numbers, 151
string, 102
strings, 151
True, 151

W

wiersz poleceń, 63, 272
argument, *Patrz:* argument wiersza poleceń
interfejsu, *Patrz:* CLI
wyjątek
obsługa, 228, 233
ValueError, 228

Z

zmienna, 40
argumentów, 64
argv, 64
globalna, 90, 150, 211
nazwa, 45
skrócona, 46
w funkcji, 88
w skrypcie, 88
znak
!=, *Patrz:* operator logiczny !=
" \t, 56
#, *Patrz:* znak kratki
%, 36, 37
%%%, 153
%c, 153
%d, 152
%e, 152
%E, 152
%f, 152
%F, 152
%g, 152
%G, 152
%i, 152
%o, 152
%r, 153
%s, 153
%u, 152
%x, 152
%X, 152
/, *Patrz:* znak ukośnika prawego
^, *Patrz:* znak wstawiania
_, *Patrz:* znak podkreślenia
{}, 44
+=, 94, 127
<=, 119
=, *Patrz:* znak równości
==, 119, *Patrz:* znak równości podwójny
>, *Patrz:* znak zachęty
>=, 119
\a, 152
ASCII, *Patrz:* ASCII
\b, *Patrz:* znak backspace
backspace, 152
cudzysłowu
podwójnego, 31, 46, 48, 51, 54, 55, 56, 152
pojedynczego, 46, 48, 51, 56, 152
\f, 152
gwiazdka, 86
kratki, 31, 34, 35
łańcuch, *Patrz:* łańcuch znaków
\n, *Patrz:* znak nowej linii
nawias
klamrowy, 62
okrągły, 62
nowej linii, 56, 152
podkreślenia, 40
powrotu karetki, 152
\r, *Patrz:* znak powrotu karetki
równości, 42, 96
podwójny, 42
\t, 152
ukośnika
lewego, 56, 58, 152, 281
prawego, 58, 281
\v, 152
wstawiania, 31
zachęty, 68, 113, 146, 278

Ż

żądanie, 246, 247
odpowiedź, 248
POST, 253

ZROZUM PYTHONA, PISZ DOBRY KOD!

Python jest dojrzałym, elastycznym i wszechstronnym językiem programowania. Nadaje się do budowy przeróżnych aplikacji, a także tworzenia programów przeznaczonych do specyficznych zastosowań, takich jak badania naukowe. Aby jednak w pełni wykorzystać te imponujące możliwości, musisz pisać dobry kod: przejrzysty, zwięzły, działający poprawnie. Niestety, nie jest łatwo nauczyć się dobrego programowania. To coś więcej niż przyswojenie zestawu poleceń i słów kluczowych. Wymaga czasu, wysiłku, sporego zaangażowania i... dobrego przewodnika na tej trudnej ścieżce.

Niniejsza książka jest właśnie takim dobrym przewodnikiem dla początkujących programistów. Napisana łatwym językiem, wciągająca, duży nacisk kładzie na analizę tworzonego kodu. Jeśli tylko skoncentrujesz się na wykonywanych zadaniach, zdobędziesz się na zaangażowanie i dokładność, zrozumienie znaczenia każdej linii programu przyjdzie łatwo. Wartościowym elementem książki są wskazówki, jak zepsuć napisany kod, a następnie go zabezpieczyć — co przygotuje Cię do unikania błędów w trakcie programowania. Dzięki tej książce zdobędziesz trzy najważniejsze umiejętności każdego programisty: czytania i pisania ze zrozumieniem, dbałości o szczegóły oraz dostrzegania różnic.

Najważniejsze zagadnienia:

- przygotowanie kompletnego środowiska programistycznego
- organizowanie, pisanie, psucie i naprawianie kodu
- programowanie obiektowe
- projektowanie programu i testowanie kodu
- podstawy budowy aplikacji internetowych i prostszych gier

ZED A. SHAW jest programistą od ponad 20 lat. Chętnie angażuje się w różne projekty open source, jest również uznanym autorem wielu książek i artykułów dotyczących technik programowania — jego publikacje są czytane i dyskutowane przez miliony czytelników na całym świecie. Posiada niezwykłą umiejętność pisania o trudnych zagadnieniach w sposób przystępny, żywy i interesujący, a równocześnie zmuszający czytelnika do myślenia. W wolnym czasie studiuje malarstwo i historię sztuki.

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
WWW.SZKOLENIA.HELION.PL
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-4141-8
9 788328 341418
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 59,00 zł
Pearson
Addison-Wesley