

O'REILLY®

Head First

Python

Rusz głową!

Przewodnik
dla uczących się
podstaw Pythona

Paul Barry

Wydanie III



Helion

Tytuł oryginału: Head First Python: A Learner's Guide to the Fundamentals of Python Programming,
A Brain-Friendly Guide, 3rd Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-289-0700-3

© 2024 Helion S.A.

Authorized Polish translation of the English edition of *Head First Python*, 3E
ISBN 9781492051299 © 2023 Paul Barry.

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.

Polish edition copyright © 2024 by Helion S.A.

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

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

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

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/pytrg3>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<https://ftp.helion.pl/przyklady/pytrg3.zip>

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

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

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści (podsumowanie)

Wprowadzenie	xxiii
0 Dlaczego Python? Podobny, ale inny	1
1 Zaczynamy	41
2 Listy liczb	77
3 Lista plików	121
4 Sformatowane literały łańcuchowe	169
5 Grunt to organizacja	215
6 Tworzenie aplikacji internetowych	247
7 Wdrażanie	303
8 Praca z HTML-em	333
9 Praca z danymi	371
9 ^{1/2} Praca ze słownikami ramkami danych	407
10 Bazy danych	431
11 Listy składane	485
12 Kolejne wdrożenie	547
Dodatek: Dziesięć najważniejszych zagadnień, o których nie napisaliśmy	575
Skorowidz	589

Spis treści (ten z prawdziwego zdarzenia)

Wprowadzenie

Twój mózg jest nastawiony na Pythona. Tutaj próbujesz się czegoś nauczyć, a Twój mózg wyświadcza Ci przysługę, upewniając się, że ta wiedza nie wchodzi Ci do głowy. Twój mózg myśli sobie: „Lepiej zostawić miejsce na ważniejsze rzeczy, takie jak to, których dzikich zwierząt należy unikać albo czy jazda nago na snowboardzie jest złym pomysłem”. Zatem jak oszukać mózg, by myślał, że Twoje życie zależy od umiejętności programowania w Pythonie?

Dla kogo jest ta książka?	xxiv
Wiemy, co myślisz	xxv
Wiemy, co myśli Twój mózg	xxv
Metapoznanie — myślenie o myśleniu	xxvii
Oto co MY zrobiliśmy	xxviii
Przeczytaj to	xxx
Instalujemy najnowszą wersję Pythona	xxxii
Sam Python to nie wszystko	xxxiii
Skonfiguruj VS Code wedle własnych upodobań	xxxiv
Dodaj dwa wymagane rozszerzenia do VS Code	xxxv
Wsparcie dla Pythona w VS Code jest nie do pobicia	xxxvi
Zespół recenzentów technicznych	xxxviii
Podziękowania	xxxix

Dlaczego Python?

**Podobny, ale inny**

Python rozpoczyna liczenie od zera, co powinno brzmieć znajomo.

W rzeczywistości Python ma **wiele wspólnego** z innymi językami programowania. Istnieją w nim **zmienne, pętle, instrukcje warunkowe, funkcje** i tym podobne. W tym, pierwszym rozdziale, zabierzemy Cię na **wycieczkę, o dość ogólnym charakterze**, po samych podstawach Pythona, prezentując ten język bez zbytecznego zagłębiania się w szczegóły. Dowiesz się, jak **tworzyć i uruchamiać** kod w notatnikach Jupyter Notebook (działających bezpośrednio w edytorze VS Code). Zobaczysz, jak wiele funkcji programistycznych, które możesz wykorzystać do **wykonywania** zadań, jest **wbudowanych** w Pythona. Dowiesz się również, że choć Python współdzieli wiele koncepcji z innymi językami programowania, sposób, w jaki przejawiają się one w kodzie Pythona, może być, no cóż, **inny**. Nie zrozum tego źle: mówimy tu o odmienności w **dobrym**, a nie *złym* sensie. Czytaj dalej, aby dowiedzieć się więcej.



Przygotowania do wykonywania kodu	7
Przygotowanie do pierwszego doświadczenia z Jupyterem	8
Wprowadźmy trochę kodu do edytora notatnika	9
Naciśnij Shift+Enter, aby uruchomić kod	10
Co zrobić, jeśli chcesz więcej niż jedną kartę?	15
Przyjrzyj się bliżej kodowi rysowania kart	17
Wielka czwórka: lista, krotka, słownik i zbiór	18
Stwórz swoją talię kart z użyciem zbioru	19
print dir — podwójne mambo	20
Uzyskiwanie pomocy na temat wyników dir	21
Wypełniamy zbiór kartami	22
Teraz to wygląda jak talia kart	24
A właściwie czym jest zmienna card?	25
Musisz coś znaleźć?	28
Zróbmy przerwę i sprawdźmy, czym dysponujemy	29
Python jest dostarczany wraz z bogatą biblioteką standardową	30
W Pythonie piszesz tylko kod, którego potrzebujesz	34
Kiedy już myślałeś, że to koniec...	39

1 Zaczynamy Wskakujemy!

Najlepszym sposobem na naukę nowego języka jest napisanie kodu.

A jeśli zamierzasz napisać jakiś kod, będziesz potrzebować **prawdziwego** problemu do rozwiązania. Na szczęście mamy taki. W tym rozdziale rozpoczniesz swoją przygodę z tworzeniem aplikacji w Pythonie od poznania naszego przyjaznego znajomego z sąsiedztwa — **trenera pływania**. Zaczniiesz od łańcuchów znaków w Pythonie, ucząc się, jak nimi manipulować, jednocześnie cały czas będziesz pracować nad rozwiązaniem problemu trenera. Dodatkowo przyjrysz się dokładniej także wbudowanej strukturze danych Pythona — **liście**, dowiesz się, jak działają **zmienne**, i nauczysz się tak czytać **komunikaty o błędach** generowane przez Pythona, żeby nie zwariować, a wszystko w ramach prac nad rozwiązaniem *prawdziwego* problemu z użyciem *prawdziwego* kodu Pythona. A zatem... wskakujemy (i to na główkę!)



Jak trener pracował do tej pory?	43
Trener potrzebuje lepszego stopera	44
Plik i arkusz kalkulacyjny są ze sobą „powiązane”	49
Nasze pierwsze zadanie: wyodrębnić dane z nazwy pliku	50
Łańcuch znaków to obiekt mający atrybuty	51
Pobieramy dane pływaka z nazwy pliku	56
Nie próbuj zgadywać, co robi dana metoda...	57
Dzielenie łańcuchów znaków na części	58
Wciąż zostaje nam coś do zrobienia	60
Komunikaty błędów czytaj od dołu do góry	64
Uważaj na łączenie wywołań metod	65
Wypróbujmy inną metodę łańcuchów znaków	67
Pozostało nam jedynie utworzenie paru zmiennych	70
Zadanie nr 1 zostało wykonane!	75
Zadanie nr 2: przetworzenie danych z pliku	76

2

Listy liczb

Przetwarzanie danych z list

Im więcej kodu piszesz, tym lepszy się stajesz. To takie proste.

W tym rozdziale będziemy kontynuować pisanie kodu w Pythonie, aby pomóc trenerowi. Dowiesz się w nim, jak **odczytywać** dane z plików dostarczonych przez trenera, wczytując ich poszczególne wiersze do **listy**, jednej z najpotężniejszych wbudowanych **struktur danych** Pythona. Oprócz tworzenia list na podstawie danych z pliku nauczysz się także tworzyć listy od podstaw i **dynamicznie** je **powiększać** w zależności od potrzeb. Będziesz także przetwarzać listy przy użyciu jednej z najpopularniejszych konstrukcji Pythona do wielokrotnego powtarzania operacji: pętli for. Zajmiesz się także konwertowaniem wartości z jednego formatu danych na inny, a nawet poznasz swojego nowego najlepszego przyjaciela. Napites się już kawy i zjadłeś ciasto, więc czas zakasać rękawy i wrócić do pracy.

Zadanie nr 2: przetworzenie danych z pliku	78
Zrób sobie kopię danych trenera	79
Wbudowana funkcja open operuje na plikach	80
Stosowanie with do otwierania (i zamykania) pliku	81
Zmienne są tworzone dynamicznie, wedle potrzeb	84
Tym, o co nam naprawdę chodzi, są dane z pliku	85
Już mamy dane pływaków z pliku	87
Kolejne czynności wyglądają znajomo	90
Wiedza z poprzedniego rozdziału zaczyna przynosić korzyści	93
Konwersja łańcucha z czasem na wartość liczbową	94
Setne sekundy w Pythonie	96
Krótką prezentacją pętli for w Pythonie	98
Pojedynek na gołe pięści: pętla for kontra pętla while	101
Idziemy do przodu, robiąc wielkie postępy!	103
Zachowajmy kopię skonwertowanych czasów	104
Wyświetlenie listy metod listy	105
Nadszedł czas, by obliczyć średnią	110
Konwersja średniej na format łańcucha z czasem	111
Pora zebrać wszystko w całość	115
Zadanie nr 2 jest (w końcu) gotowe!	118

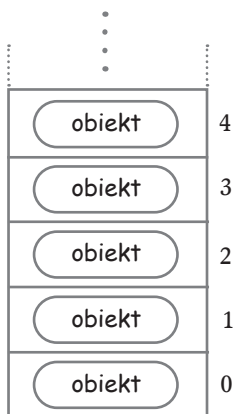
3

Lista plików

Funkcje, moduły i pliki

Twój kod nie może wiecznie żyć w notatniku. Chce być wolny.

A jeśli chodzi o uwolnienie kodu i **udostępnianie** go innym, pierwszym krokiem jest wspomniana już wcześniej **funkcja**, a tuż za nią **moduł**, który pozwala organizować i udostępniać kod. W tym rozdziale utworzysz funkcję bezpośrednio na podstawie kodu, który już napisałeś, oraz moduł, który **będzie można udostępnić**. Zaraz potem zastosujesz swój moduł w pracach nad przetwarzaniem danych trenera, w których wykorzystasz dodatkowo pętle `for`, instrukcje `if`, warunki i kolejne funkcje z **biblioteki standardowej** Pythona (PSL). Dowiesz się także, jak **komentować** swoje funkcje (co zawsze jest godne polecenia). Jest wiele do zrobienia, więc do dzieła!



Lista

Już masz większość potrzebnego kodu	123
Jak utworzyć funkcję w Pythonie?	124
Zapisuj swój kod tak często, jak masz ochotę	125
Zwyczajne skopiowanie kodu nie wystarczy	126
Konieczniesz skopiuj cały niezbędny kod	127
Używaj modułów do współdzielenia kodu	134
Pław się w blasku zwróconych danych	135
W razie konieczności funkcje mogą zwracać krotki	137
Przygotujmy listę nazw plików trenera	143
Nadszedł czas, by pobawić się w detektywa...	144
Co można robić z listami?	145
Czy przyczyną problemu są dane, czy nasz kod?	153
Decyzje, decyzje i jeszcze raz decyzje	157
Poszukajmy dwukropka „w” łańcuchu	158
Czy w efekcie przetworzyłeś 60 plików?	165
Kod dla trenera zaczyna nabierać kształtu	166

4

Sformatowane literały łańcuchowe

Twórz wykresy na podstawie danych

Czasami to najprostsze metody pozwalają wykonać zadanie.

W tym rozdziale zajmiemy się w końcu tworzeniem wykresów słupkowych dla trenera. Zrobimy to, używając wyłącznie łańcuchów znaków. Już wiesz, że łańcuchy znaków w Pythonie są pełne **wbudowanych dobroci**, a **sformatowane literały łańcuchowe** Pythona, znane również jako *f-łańcuchy*, zwiększają jeszcze te możliwości w dość zgrabny sposób. Może się wydawać dziwne, że proponujemy tworzenie wykresów słupkowych przy użyciu tekstu, ale jak zaraz się przekonasz, nie jest to tak absurdalne, jak mogłoby się wydawać. Po drodze użyjesz Pythona do tworzenia **plików**, a także uruchomisz przeglądarkę internetową za pomocą zaledwie kilku wierszy kodu. A pod koniec rozdziału życzenie trenera zostanie spełnione: wykresy słupkowe będą **automatycznie generowane** na podstawie danych z zarejestrowanymi czasami pływania. Zatem: do dzieła!

Eliza (poniżej 14 lat), 100m, styl: grzbietowy



Średni czas: 1:29,20

Tworzymy prosty wykres słupkowy przy użyciu HTML-a i SVG	174
Przechodzimy od prostego wykresu do wykresu dla trenera	177
Konstruujemy potrzebne łańcuchy z HTML-em w kodzie	178
Konkatenacja łańcuchów znaków nie jest skalowalna	181
F-łańcuchy są bardzo popularną możliwością Pythona	186
Dzięki użyciu f-łańcuchów generowanie kodu SVG jest proste!	187
Wciąż dysponujemy danymi, prawda?	188
Zadbaj, by zwracane były wszystkie niezbędne dane	189
Teraz już mamy dane liczbowe, ale czy nadają się one do użycia?	190
Pozostaje nam już jedynie dokończyć stronę WWW	199
Zapisywanie w plikach, podobnie jak odczyt z plików, jest bezbolesne	200
Zostały już tylko poprawki estetyczne	204
Czas na kolejną niestandardową funkcję	206
Dodajmy kolejną funkcję do naszego modułu	207
Co jest z tymi wartościami setnych sekundy?	210
Zaokrąglanie nie jest tym, czego chcemy (w tym przypadku)	211
Wszystko idzie doskonale...	213

5

Grunt to organizacja

Decyzje dotyczące struktur danych**Twój kod musi gdzieś umieścić swoje dane w pamięci.**

A jeśli chodzi o organizowanie danych gdzieś... **w pamięci**, wybór struktury danych do użycia może być krytyczny i często stanowi różnicę między niechlujnym rozwiązaniem, które *działa*, a **eleganckim** rozwiązaniem, które *działa dobrze*. W tym rozdziale poznasz kolejną z wbudowanych struktur danych Pythona — **słownik**, który jest często łączony z wszechobecną listą w celu tworzenia **złożonych** struktur danych. Trener potrzebuje łatwego sposobu na wybranie danych dowolnego pływaka, a kiedy umieścimy dane trenera w słowniku, wyszukiwanie stanie się dziecinnie proste!

⋮	⋮
klucz#4	obiekt
klucz#1	obiekt
klucz#3	obiekt
klucz#2	obiekt

Słownik

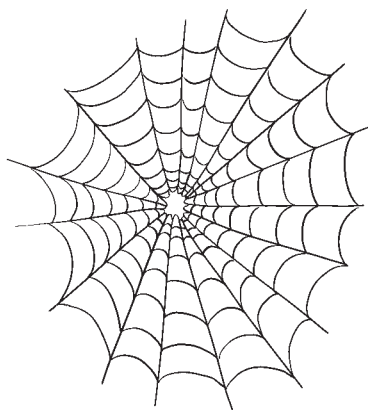
Pobierzmy listę imion pływaków	217
Lista-zbiór-lista — sztuczka usuwająca powtórzenia	219
Trener dysponuje już listą imion	221
Mała zmiana robi „wielką” różnicę	222
Każda krotka jest unikatowa	223
Bardzo szybkie wyszukiwanie z użyciem słowników	226
Słowniki to magazyny par klucz-wartość ułatwiające wyszukiwanie	227
Anatomia tworzenia słownika	230
Słowniki są zoptymalizowane pod kątem szybkości wyszukiwania	238
Wyświetlanie całego słownika	239
Moduł pprint wyświetla dane w atrakcyjny sposób	240
Twój słownik list jest łatwy do przetwarzania	241
Wszystko zaczyna do siebie pasować	242

6

Tworzenie aplikacji internetowej

Pisanie aplikacji internetowych**Zapytaj dziesięciu programistów, którego frameworka użyć...**

...i prawdopodobnie otrzymasz jedenaście sprzecznych odpowiedzi! 😊 Jeśli chodzi o tworzenie stron WWW, Python nie jest pozbawiony możliwości *wyboru* technologii, z których każda ma lojalną i wspierającą społeczność programistów. W tym rozdziale zajmiemy się **tworzeniem stron WWW** i błyskawicznie przygotujemy aplikację internetową dla trenera, która będzie wyświetlać wykresy słupkowe na podstawie danych dowolnego pływaka. Po drodze dowiesz się, jak korzystać z **szablonów** HTML, funkcji **dekoratorów**, metod HTTP get i set i nie tylko. Nie ma czasu do stracenia: trener chce pochwalić się swoim nowym systemem. Do dzieła!



Instalujemy Flaska z PyPI	249
Przygotuj katalog aplikacji	250
Podczas pracy z kodem mamy różne możliwości	253
Anatomia minimalnej aplikacji Flaska	255
Buďujemy aplikację internetową kawałek po kawałku	262
O co chodzi z tym NameError?	267
Flask udostępnia wbudowaną obsługę sesji	269
Do obsługi sesji Flask używa słownika	270
Wprowadzenie w kodzie „lepszej poprawki”	273
Tworzenie szablonów Jinja2 oszczędza czas	279
Rozszerz szablon base.html, by utworzyć więcej stron	281
Dynamiczne tworzenie listy rozwijanej	284
Musimy w jakiś sposób przetworzyć dane formularza	289
Dane z formularza są dostępne jako słownik	290
Funkcje obsługują domyślne wartości parametrów	295
Domyślne wartości parametrów są opcjonalne	296
Końcowa wersja kodu, strona 1. z 2	297
Końcowa wersja kodu, strona 2. z 2	298
Jak na pierwszą aplikację internetową wygląda to dobrze	300
System dla trenera jest gotowy do użycia	301

7

Wdrażanie

Uruchamiaj swój kod gdziekolwiek**Uruchomienie kodu na własnym komputerze to jedno...**

Tym, czego naprawdę chcesz, jest **wdrożenie** swojego kodu tak, aby użytkownicy mogli go łatwo uruchamiać. A jeśli możesz to zrobić *bez* większego zamieszania, tym lepiej. W tym rozdziale dokończysz aplikację internetową trenera, dodając do niej trochę **stylu**, a następnie **wdrożysz** ją w **chmurze**. Nie myśl jednak, że jest to coś, co zwiększa złożoność do 11 — nic takiego się nie dzieje. W poprzedniej edycji tej książki chwalono się, że wdrożenie zajmuje „około 10 minut”, i nadal jest to szybka robota... chociaż w tym rozdziale wdrożysz swoją aplikację internetową w 10 krokach. Chmura już czeka, by hostować aplikację internetową trenera. A zatem wdrażamy!



Wciąż coś wydaje się nie w porządku	311
Jinja2 wykonuje kod zapisany pomiędzy <code>{{ i }}</code>	316
Dziesięć kroków wdrażania aplikacji w chmurze	318
Konto dla początkujących to wszystko, czego Ci trzeba	319
Nic Cię nie powstrzyma przed natychmiastowym rozpoczęciem...	320
W razie wątpliwości pozostań przy ustawieniach domyślnych	321
Wygenerowana aplikacja nie robi wiele	322
Wdrażanie aplikacji w serwisie PythonAnywhere	323
Rozpakuj swój kod z poziomu konsoli	324
Skonfiguruj kartę Web, aby wskazywała na Twój kod	325
Zmodyfikuj plik WSGI swojej aplikacji	326
Twoja aplikacja działająca w chmurze jest gotowa!	330

8

Praca z HTML-em

Webscraping

W idealnym świecie łatwo byłoby uzyskać dostęp do wszystkich potrzebnych danych.

Niestety w rzeczywistości rzadko kiedy tak się dzieje. Przykład: dane są publikowane w sieci. Dane osadzone w kodzie HTML są przeznaczone do **wyświetlania** przez przeglądarki internetowe i **odczytywania** przez ludzi. Ale co, jeśli musisz **przetwarzać** dane osadzone na stronach WWW za pomocą kodu? Nie masz szczęścia? Cóż, na szczęście Python jest w pewnym sensie gwiazdą, jeśli chodzi o **webscraping**, czyli pobieranie danych ze stron WWW, a w tym rozdziale dowiesz się, jak to robić. Dowiesz się również, jak **analizować** te strony HTML, aby wyodrębnić z nich użyteczne dane. Po drodze spotkasz **wycinki** i **zupę**. Ale nie martw się, to wciąż jest *Python. Rusz głową, a nie Gotowanie. Rusz głową...*



Trener potrzebuje więcej danych	334
Przed pozyskaniem danych ze stron WWW musisz je poznać	336
Potrzebujemy planu działania	337
Przewodnik po scrapowaniu stron WWW krok po kroku	338
Czas na poznanie technologii parsowania HTML-a	340
Pobieramy nieprzetworzoną stronę HTML z Wikipedii	343
Poznaj dane, które chcesz scrapować	344
Wycinek można skopiować z dowolnej sekwencji	346
Anatomia wycinków, 1. z 3	347
Anatomia wycinków, 2. z 3	348
Anatomia wycinków, 3. z 3	349
Najwyższy czas na nieco poważniejsze parsowanie HTML-a	354
Przeszukiwanie zupy w celu znalezienia interesujących nas znaczników	355
Zwróconą zupę także można przeszukiwać	356
Która tabela zawiera potrzebne dane?	359
Cztery duże tabele i cztery zestawy rekordów świata	361
Czas zająć się pobraniem faktycznych danych	362
Pobieramy dane ze wszystkich tabel, 1. z 2	366
Pobieramy dane ze wszystkich tabel, 2. z 2	367
Ta zagnieżdżona pętla jest podchwytliwa	370

9

Praca z danymi

Manipulowanie danymi**Czasami nasze dane nie są uporządkowane tak, jak powinny.**

Być może masz *listę list*, ale tak naprawdę potrzebujesz *słownika słowników*. A może musisz powiązać wartość w jednej strukturze danych z wartością w innej strukturze danych, ale wartości te nie do końca do siebie pasują. To takie frustrujące. Nie martw się: moc Pythona jest dostępna i czeka, by Ci pomóc. W tym rozdziale użyjesz Pythona do **manipulowania** danymi, aby zmienić dane uzyskane pod koniec poprzedniego rozdziału w coś naprawdę *użytecznego*. Zobaczysz, jak wykorzystać słownik Pythona jako tablicę przeglądową (ang. *lookup table*). W tym rozdziale wykorzystamy konwersję, integrację, aktualizację, wdrożenia i wiele więcej. A na końcu tego wszystkiego serwetkowa specyfikacja trenera stanie się *rzeczywistością*. Nie możesz się doczekać? My też... zatem do dzieła!



Naginanie danych do naszej woli...	372
Teraz już dysponujesz niezbędnymi danymi...	376
Skorzystaj z tego, co już potrafisz...	378
Czy tu aby nie jest za dużo danych?	381
Odrzucenie danych sztafet	382
Teraz jesteś już gotów, by zaktualizować stronę z wykresami słupkowymi	383
Python posiada wbudowaną bibliotekę do obsługi JSON-a	385
JSON jest formatem tekstowym, ale niezbyt ładnym	386
Dochodzimy do integracji z aplikacją internetową	390
Wszystko, co trzeba: skopiuj i wklej...	391
Dodawanie rekordów świata do wykresu słupkowego	392
Czy najnowsza wersja aplikacji internetowej jest gotowa?	396
Serwis PythonAnywhere zadba o to...	400
Musisz skopiować także swój kod pomocniczy	401
Prześlij aplikację do serwisu PythonAnywhere	402
Poinstruj serwis, by zaczął wykonywać Twój najnowszy kod	403
Przetestuj skrypt przed wdrożeniem go w chmurze	404
Niech zadanie będzie wykonywane codziennie o 1:00 w nocy	405

9¹/₂Praca ze słowami i ramkami danych
Dane tabelaryczne

Czasami wygląda to tak, jakby wszystkie dane świata chciały znaleźć się w tabeli.

Dane tabelaryczne są *wszędzie*. Rekordy świata w pływaniu z poprzedniego rozdziału to dane **tabelaryczne**. Jeśli masz na tyle lat, żeby pamiętać książki telefoniczne, to były one przykładem danych tabelarycznych. Wyciągi bankowe, faktury, arkusze kalkulacyjne: zgadłeś, to wszystko są właśnie dane tabelaryczne. W tym *krótkim* rozdziale dowiesz się nieco o jednej z najpopularniejszych bibliotek do analizy danych tabelarycznych w Pythonie: bibliotece *pandas*. To będzie jedynie pobieżna prezentacja tego, co potrafi ta biblioteka, ale dowiesz się z niej wystarczająco dużo, aby nauczyć się wykorzystywać najczęściej używaną strukturę danych *pandas*, ramkę danych (ang. *dataframe*), gdy następnym razem będziesz musiał przetwarzać fragment danych tabelarycznych.



Słoń w pomieszczeniu... a może to panda?	408
Słownik słowników w <i>pandas</i> ?	409
Zaczynamy zgodnie z przyjętymi konwencjami	410
Lista ramek danych	411
Wybór kolumn z ramki danych	412
Ramka danych na słownik, podejście 1.	413
Usuwanie niepożądanych danych z ramki danych	414
Negowanie wyrażeń warunkowych w <i>pandas</i>	415
Ramka danych na słownik, podejście 2.	416
Ramka danych na słownik, podejście 3.	417
To jest kolejny słownik słowników	418
Polonizacja słownika rekordów	419
Gdzie są nasze dane?	420
Porównanie <i>gazpacho</i> i <i>pandas</i>	424
To był jedynie błyskawiczny rzut oka...	430

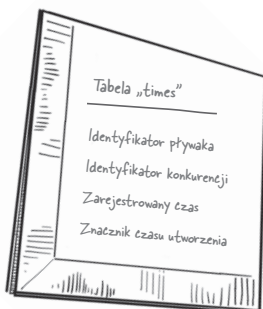
10

Bazy danych

Organizujemy się

Prędzej czy później pojawi się konieczność zarządzania danymi aplikacji.

A kiedy trzeba bardziej odpowiednio **zarządzać** danymi, Python (sam w sobie) może nie wystarczyć. W takiej sytuacji będziesz musiał sięgnąć po swój ulubiony silnik **bazy danych**. Aby wszystko było... em, eh... *łatwe do zarządzania*, będziemy korzystali z silników baz danych, które obsługują niezawodny stary **SQL**. W tym rozdziale nie tylko **utworzysz** bazę danych, a następnie dodasz do niej kilka **tabel**, ale także **wstawisz** dane do bazy danych, **wyberzesz** i **usuniesz** je z niej, wykonując wszystkie te zadania z poziomu kodu Pythona za pomocą odpowiednich zapytań SQL.



Odezwał się do nas trener...	432
Planowanie popłaca...	435
Zadanie nr 1: określenie struktury bazy danych	437
Struktura z chusteczek + dane	439
Instalowanie modułu DBcm z PyPI	440
Początki z DBcm i ze SQLite	441
DBcm świetnie współdziała z instrukcją „with”	442
Kod SQL zapisuj pomiędzy sekwencjami trzech cudzysłowów	444
Nie wszystkie polecenia SQL zwracają wyniki	446
Twoje tabele są gotowe (a zadanie nr 1 jest wykonane)	451
Określanie listy plików z wynikami pływaków	452
Zadanie nr 2: dodawanie danych do tabeli	453
Zachowaj bezpieczeństwo, używając w SQL-u symboli zastępczych Pythona	455
Powtórzmy ten proces dla tabeli konkurencji	470
Pozostaje już jedynie tabela czasów...	474
Czasy są zapisane w plikach pływaków...	475
Narzędzie do aktualizacji bazy danych, 1. z 2	481
Narzędzie do aktualizacji bazy danych, 2. z 2	482
Zadanie nr 2 (w końcu) zostało wykonane	483

11

Listy składane

Integracja z bazą danych**Po przygotowaniu tabel bazy danych nadszedł czas na integrację.**

Twoja aplikacja internetowa może zyskać **elastyczność** wymaganą przez trenera poprzez wykorzystanie zestawów danych zapisywanych w tabelach bazy danych, a w tym rozdziale utworzysz moduł narzędzi, który pozwoli Twojej aplikacji internetowej **korzystać** z silnika bazy danych. W niekończącym się dążeniu do robienia więcej za pomocą mniejszej ilości kodu nauczysz się czytać i pisać **listy składane**, które są prawdziwą supermocą Pythona. Będziesz także **ponownie używać** wielu fragmentów już istniejącego kodu na nowe i interesujące sposoby. Zatem zaczynamy. Przed nami wiele pracy związanej z **integracją**.

Wypróbujemy zapytania w nowym notatniku	488
Pięć wierszy kodu pętli staje się jednym	491
Podwójne mambo bez podkreśleń	493
Jedno zapytanie z głowy, pozostają trzy...	498
Dwa zapytania z głowy, pozostają jeszcze dwa...	500
Ostatnie (zapytanie), choć nie najmniej ważne...	501
Kod narzędzi danych, część 1. z 2	506
Kod narzędzi danych, część 2. z 2	507
Już niemal nadszedł czas na integrację bazy danych	510
Aktualizacja istniejącego kodu aplikacji internetowej	522
Zatem... o co chodzi z tym szablonem?	526
Zajmijmy się wyświetleniem konkurencji	530
Pozostaje nam już jedynie wyświetlenie wykresu	534
Przełóżmy ostatni kod modułu klubplywacki.py	536
Poznaj szablon Jinja2 do generowania kodu SVG	538
Moduł narzędzia_konwersji	540
list zip... że co?!?	543
Integracja bazy danych z aplikacją została zakończona!	545

```
[sql for sql in dir(zapytania) if not sql.startswith("__")]
```


12

Kolejne wdrożenie
Ostatnie szlify**Po przygotowaniu tabel bazy danych nadszedł czas na integrację.**

W tym, ostatnim rozdziale tej książki dostosujesz aplikację internetową dla trenera do korzystania z **MariaDB** jako serwera bazy danych, a nie, jak dotychczas, ze SQLite; oprócz tego zrobisz wszystko, co konieczne, by wdrożyć najnowszą wersję aplikacji w serwisie PythonAnywhere. W ten sposób trener uzyska dostęp do systemu, który obsługuje **dowolną** liczbę pływaków uczestniczących w **dowolnej** liczbie sesji treningowych. W tym rozdziale nie znajdziesz wielu nowych informacji o programowaniu w Pythonie, ponieważ większość czasu poświęcimy na dostosowywanie istniejącego kodu do korzystania z MariaDB i działania w serwisie *PythonAnywhere*. Kod Pythona nigdy nie istnieje w **izolacji**: wchodzi w **interakcje** ze środowiskiem, w którym działa, i systemami, od których zależy.

```
mysql> select count(*) from events;
+-----+
| count(*) |
+-----+
|         14 |
+-----+
1 row in set (0.01 sec)

mysql> select count(*) from swimmers;
+-----+
| count(*) |
+-----+
|         23 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from times;
+-----+
| count(*) |
+-----+
|        467 |
+-----+
1 row in set (0.02 sec)
```

Migracja do MariaDB	551
Przenoszenie danych trenera do MariaDB	552
Ponowne wykorzystanie tabel, 1. z 2	552
Wprowadź trzy zmiany w pliku schema.sql	553
Ponowne wykorzystanie tabel, 2. z 2	554
Sprawdź, czy tabele zostały zdefiniowane poprawnie	555
Skopiowanie istniejących danych do bazy MariaDB	556
Zadbaj o zgodność zapytań z MariaDB	558
Zmienić trzeba także kod narzędzi bazy danych	559
Utwórz bazę danych w serwisie PythonAnywhere	562
Popraw słownik z informacjami do nawiązania połączenia z bazą	563
Kopiowanie wszystkiego do chmury	564
Zaktualizuj swoją aplikację, wgrywając najnowszy kod	565
Jeszcze tylko kilka kroków...	566
Wypełnij swoją bazę danych w chmurze danymi	567
Nadszedł czas na „Jazdę próbną” aplikacji w chmurze	568
Czy jest jakiś problem z PythonAnywhere?	570
Trener jest naprawdę szczęśliwy!	572

Dodatek. Pozostałości

Dziesięć najważniejszych zagadnień, o których nie napisaliśmy

Głęboko wierzymy, że bardzo ważne jest wiedzieć, kiedy skończyć.

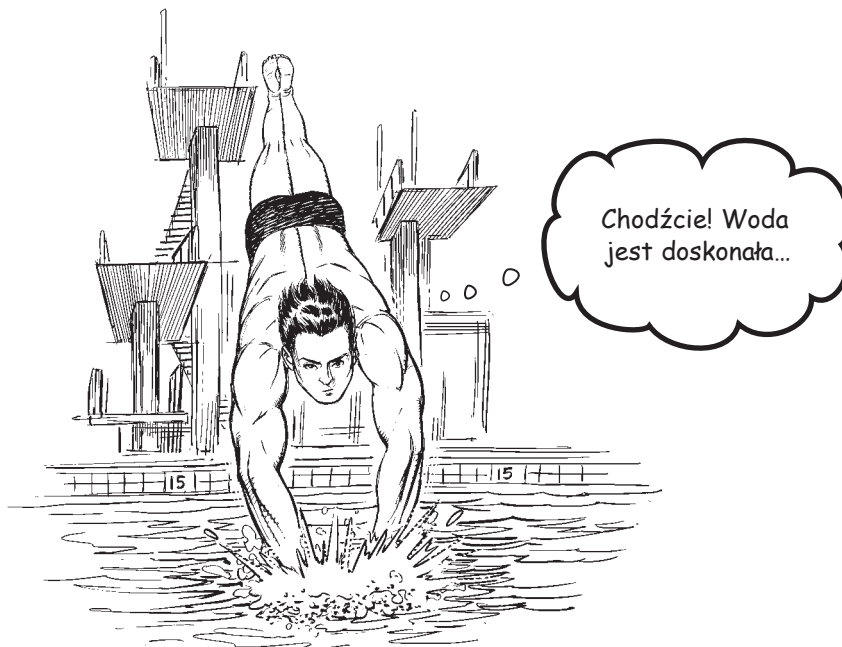
Zwłaszcza gdy autor pochodzi z *Irlandii*, narodu słynącego z osób, które nie do końca wiedzą, kiedy przestać mówić 😊 A my uwielbiamy rozmawiać o Pythonie, naszym ulubionym języku programowania. W tym „Dodatku” przedstawiamy dziesięć najważniejszych zagadnień, o których z radością byśmy Ci opowiedzieli, gdybyśmy mieli do dyspozycji kolejnych czterysta stron. Są tu informacje o klasach, obsłudze wyjątków, testowaniu, morsie (serio? o *morsie*? tak: o morsie), instrukcjach swi tch, dekoratorach, menedżerach kontekstu, współbieżności, podpowiedziach typów, środowiskach wirtualnych i narzędziach dla programistów. Jak powiedzieliśmy, zawsze jest jeszcze więcej do omówienia. Zatem śmiało, przewróć kartkę i ciesz się kolejnymi kilkunastoma stronami!

1. Klasy	576
2. Wyjątki	579
3. Testowanie	580
4. Wyrażenie przypisania	581
5. Gdzie jest switch? Jaki switch?	582
6. Zaawansowane możliwości języka	583
7. Współbieżność	584
8. Podpowiedzi typów	585
9. Środowiska wirtualne	586
10. Narzędzia	587
Skorowidz	589



1. Zaczynamy

Wskakujemy!



Najlepszym sposobem na naukę nowego języka jest napisanie kodu.

A jeśli zamierzasz napisać jakiś kod, będziesz potrzebować **prawdziwego** problemu do rozwiązania. Na szczęście mamy taki. W tym rozdziale rozpoczniesz swoją przygodę z tworzeniem aplikacji w Pythonie od poznania naszego przyjaznego znajomego z sąsiedztwa — **trenera pływania**. Zacznieś od łańcuchów znaków w Pythonie, ucząc się, jak nimi manipulować, jednocześnie cały czas będziesz pracować nad rozwiązaniem problemu trenera. Dodatkowo przyjrzyś się dokładniej także wbudowanej strukturze danych Pythona — **liście**, dowiesz się, jak działają **zmienne** i nauczysz się tak czytać **komunikaty o błędach** generowane przez Pythona, żeby nie zwariować, a wszystko w ramach prac nad rozwiązaniem *prawdziwego* problemu z użyciem *prawdziwego* kodu Pythona. A zatem... wskakujemy (i to na główkę!)...

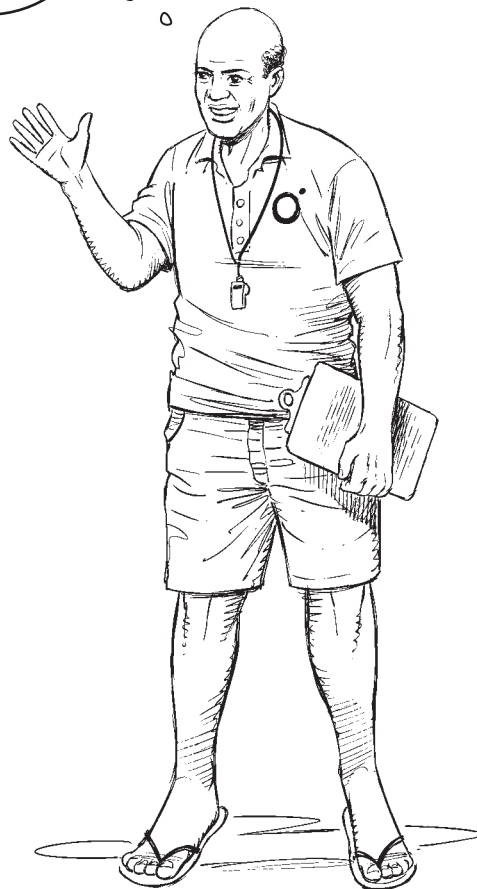
Cześć. Powiedziano mi, że jesteś osobą, z którą można porozmawiać o kodowaniu. Obecnie zmagam się z moimi arkuszami kalkulacyjnymi i zastanawiam się, czy mógłbyś pomóc mi zastąpić je czymś, czego używanie zajmuje mniej czasu? Nie mogę zwiększyć liczby członków mojego klubu pływackiego, dopóki nie rozwiążę tego problemu...

To brzmi interesująco.

W czasie Twojej późniejszej rozmowy z trenerem zaczynają ujawniać się niektóre szczegóły...

Jeśli chodzi o monitorowanie postępów młodych pływaków, trener wszystko robi *ręcznie*. Podczas sesji treningowej zapisuje czasy uzyskiwane przez poszczególnych pływaków na kartkach, a następnie w domu *ręcznie* wpisuje czasy każdego pływaka do swojego ulubionego arkusza kalkulacyjnego, aby przeprowadzić prostą analizę wyników.

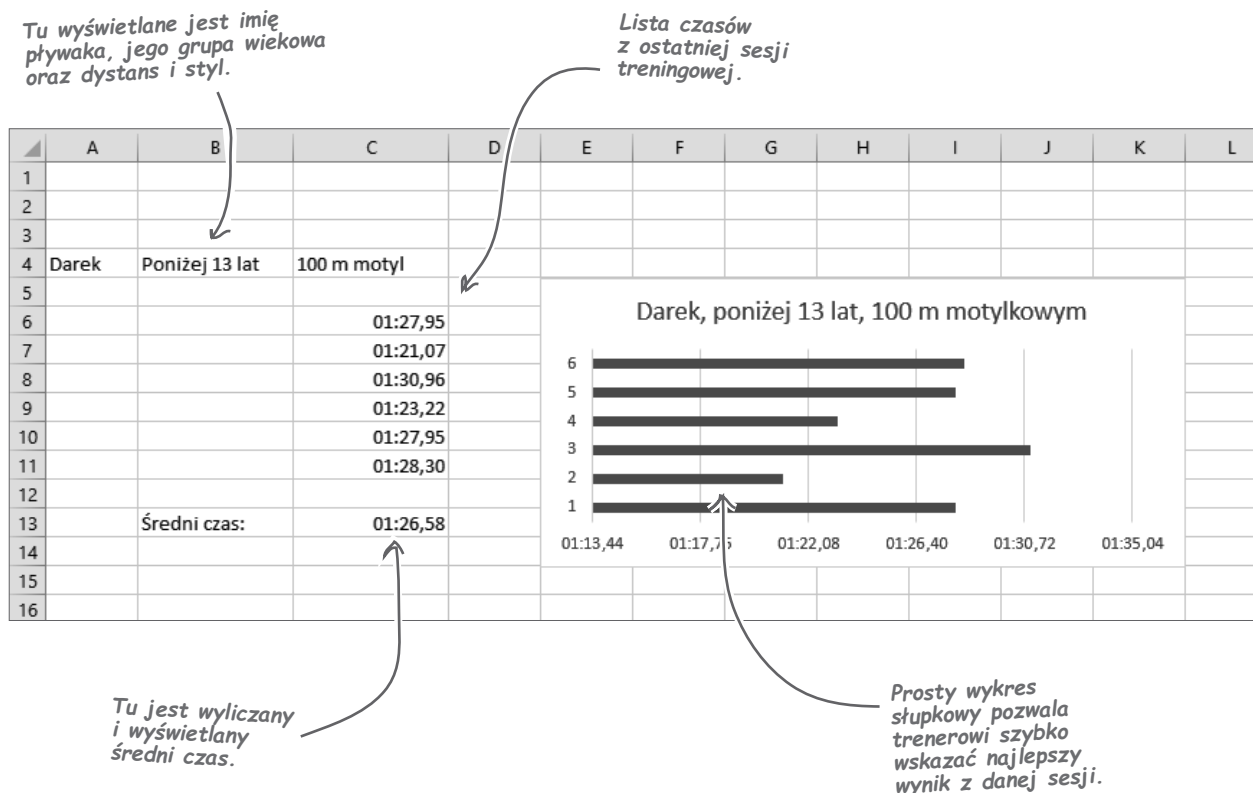
Na razie ta analiza jest nieskomplikowana. Czasy z każdej sesji treningowej są uśredniane, a prosty wykres słupkowy pozwala na szybkie wizualne sprawdzenie wyników danego pływaka. Trener nie obawia się przyznać do bycia komputerowym neofitą i otwarcie mówi, że jest o wiele lepszy w trenowaniu młodych pływaków niż „majstrowaniu przy arkuszach kalkulacyjnych”.



Jak trener pracował do tej pory?

W tej chwili trener nie może zwiększyć liczby sportowców w swoim klubie pływackim, ponieważ jego „koszty administracyjne” są zbyt duże.

Aby zrozumieć, dlaczego tak się dzieje, spójrz na jeden z arkuszy kalkulacyjnych trenera prezentujący wyniki *Darka*, który ma obecnie 12 lat, więc pływa w grupie wiekowej poniżej 13 lat:



Na pierwszy rzut oka nie wygląda to źle. Dopóki nie weźmie się pod uwagę, że po *każdej* sesji treningowej trener ma do przygotowania ponad 60 takich arkuszy kalkulacyjnych.

W klubie trenuje tylko 22 pływaków, ale ponieważ każdy z nich może pływać na różnych dystansach i różnymi stylami, tych 22 pływaków bardzo łatwo zamienia się w ponad 60 arkuszy kalkulacyjnych. Jest dużo ręcznej roboty z arkuszami.

Jak można sobie wyobrazić, cały ten proces jest *boleśnie* powolny...

Trener potrzebuje lepszego stopera

Zanim zagłębimy się w problem trenera, musimy znaleźć lepszy sposób rejestrowania czasów na sesjach treningowych, abyśmy mieli łatwiejszy dostęp do danych, gdyż używanie do tego celu kartek po prostu nie wystarczy. Szybkie przeszukanie największego na świecie sklepu internetowego pozwala znaleźć nowe urządzenie, reklamowane jako *inteligentny cyfrowy stoper podłączony do internetu*. Jak to bywa z nazwami produktów, jest to trochę kwiecista nazwa, ale inteligentny stoper pozwala trenerowi rejestrować czasy zidentyfikowanego pływaka, które są następnie przesyłane do chmury jako plik CSV, zapisany z rozszerzeniem *.txt* (żeby było ciekawiej).



W ramach przykładu poniżej przedstawiliśmy zawartość jednego z takich plików, zawierającego dane pasujące do arkusza kalkulacyjnego Darka, pokazanego na poprzedniej stronie:

Nazwa pliku zawiera imię pływaka, jego grupę wiekową, dystans i styl.

To jest plik danych wyświetlony w VS Code.

Uwaga: jest to plik tekstowy, więc czasy wyglądają jak liczby, ale nimi nie są. Są łańcuchami znaków.

```
Darek-13-100m-motyl.txt
1 1:27,95;1:21,07;1:30,96;1:23,22;1:27,95;1:28,30
2
```

Ten drugi wiersz zawsze jest pusty, więc można go bezpiecznie zignorować.



Uwaga!

Poświęć chwilę na uważne przejrzanie danych na tej stronie.

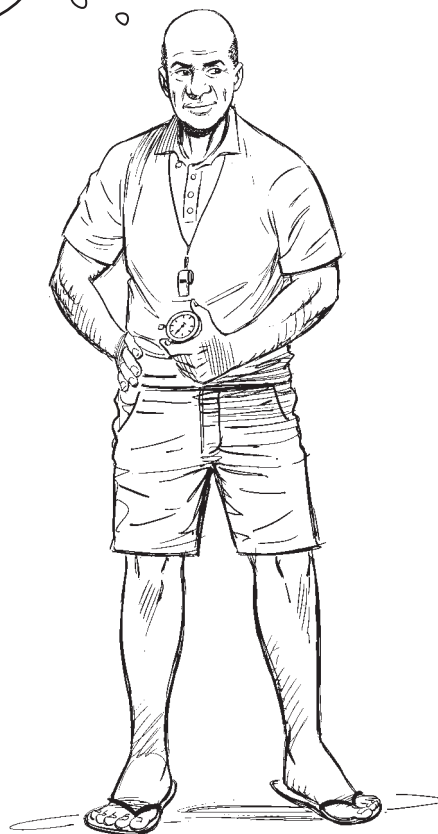
W szczególności zwróć uwagę na to, że informacje o pływaku są umieszczone w nazwie pliku, podczas gdy zawartością pliku jest lista czasów uzyskanych przez danego pływaka. Obie te dane są ważne i muszą zostać przetworzone.

Mój nowy inteligentny stoper dla każdego pływaka tworzy mały plik tekstowy. Muszę automatycznie przetworzyć każdy taki plik, wyodrębnić z niego dane pływaka, a następnie narysować wykres, który obecnie tworzę ręcznie za pomocą arkusza kalkulacyjnego. Czy da się to zrobić?

Cóż... z pewnością możemy spróbować.

Jeśli uda nam się opracować sposób przetwarzania jednego pliku, możemy go powtórzyć dla dowolnej liczby podobnie sformatowanych plików.

Rodzi się tylko pytanie: *od czego zacząć?*



Trener uwielbia swój nowy stoper i planuje używać go podczas wszystkich kolejnych sesji treningowych.

Rozmowa w boksie

Marta: Okej, ludzie, zaproponujmy kilka sugestii, jak najlepiej przetwarzać pliki generowane przez stoper, abyśmy mogli wyodrębnić z niego potrzebne nam dane.

Jan: Myślę, że to zadanie składa się z dwóch części, prawda?

Olek: Jak to?

Jan: Cóż, po pierwsze, w nazwie pliku są zapisane przydatne dane, które należy przetworzyć. Po drugie, w samym pliku znajdują się uzyskane czasy, które również należy pobrać, przekonwertować i przetworzyć.

Marta: Co masz na myśli, mówiąc o „konwertowaniu”?

Olek: Ja też chciałem o to zapytać.

Jan: Sprawdziłem z trenerem i „1:27,95” oznacza jedną minutę, 27 sekund i 95 setnych sekundy. Należy to wziąć pod uwagę podczas pracy z tymi wartościami, zwłaszcza przy obliczaniu średnich. Potrzebna jest więc jakaś konwersja wartości. Pamiętaj też, że dane w pliku są zapisywane *tekstowo*.

Olek: Dodam zatem „konwersję” do listy rzeczy do zrobienia.

Marta: I zgaduję, że nazwa pliku musi zostać w jakiś sposób podzielona na części, abyśmy uzyskali szczegółowe informacje o pływaku?

Jan: Tak. Nazwę pliku, „Darek-13-100m-motył” można podzielić w miejscach występowania znaku „-”, co da nam imię pływaka (Darek), jego grupę wiekową (poniżej 13 lat), dystans (100 m) i styl (motyl).

Olek: Oczywiście zakładając, że możemy jakoś odczytać nazwę pliku?

Jan: Czy to nie jest oczywiste?

Marta: Nie do końca, więc nadal będziemy musieli zadbać o to w kodzie, chociaż jestem pewna, że biblioteka standardowa Pythona może nam w tym pomóc.

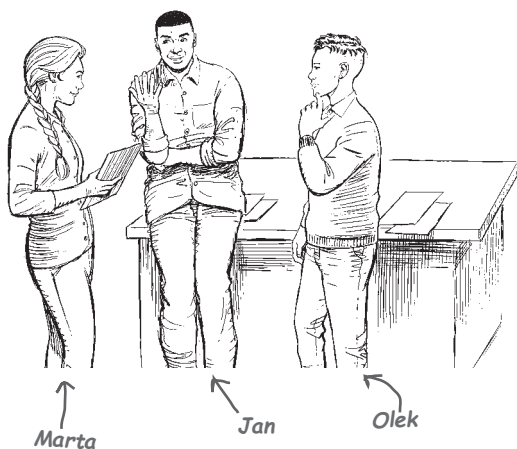
Olek: To staje się trochę skomplikowane...

Jan: Nie, jeśli zajmiemy się tym krok po kroku.

Marta: Potrzebujemy tylko planu działania.

Olek: Jeśli zamierzamy wykonać całą tę pracę w Pythonie, będziemy musieli się jeszcze trochę poduczyć.

Jan: Tak, oczywiście. Ale zanim przystąpimy do nauki, ustalmy, co musimy zrobić.





Zaostrz ołówek

Z rozmowy na poprzedniej stronie wynika, że na tym etapie zidentyfikowano dwa główne zadania: (1) wyodrębnienie danych z nazwy pliku i (2) przetworzenie danych z czasami pływania, zapisanych w pliku.

Weź ołówek i dla każdego z tych zidentyfikowanych zadań zapisz poniżej, jakie czynności są konieczne do jego rozwiązania. Nasze odpowiedzi znajdziesz na następnej stronie.

1 Wyodrębnij dane z nazwy pliku.

2 Przetwórz dane zapisane w pliku.

→ Odpowiedź znajdziesz na stronie 48.



Zaostrz ołówek

Rozwiązanie ze strony 47.

Z rozmowy na poprzedniej stronie wynika, że na tym etapie zidentyfikowano dwa główne zadania: (1) wyodrębnienie danych z nazwy pliku i (2) przetworzenie danych z czasami pływania, zapisanych w pliku.

Miałeś wziąć ołówek i dla każdego z tych zidentyfikowanych zadań zapisać, jakie czynności są konieczne do jego rozwiązania. Oto co my wymyśliliśmy. A Tobie jak poszło?

1 Wyodrębnij dane z nazwy pliku.

a) Pobierz nazwę pliku.

b) Podziel nazwę pliku w miejscach wystąpienia znaku „-”.

c) Zapisz imię pływaka, jego grupę wiekową, dystans oraz styl

w zmiennych (by można ich było używać w dalszej części kodu).

2 Przetwórz dane zapisane w pliku.

a) Odczytaj wiersze tekstu z pliku.

b) Zignoruj drugi wiersz tekstu.

c) Podziel pierwszy wiersz tekstu w miejscach wystąpienia znaku

„:” w celu uzyskania listy czasów.

d) Przekształć każdy z czasów z formatu tekstowego

„minuty:sekundy,setnesek” na liczbę.

e) Oblicz średnią wszystkich czasów i skonwertuj ją na format

„minuty:sekundy,setnesek” (w celu wyświetlenia).

f) Wyświetl zmienne z zadania nr 1, a następnie listę czasów

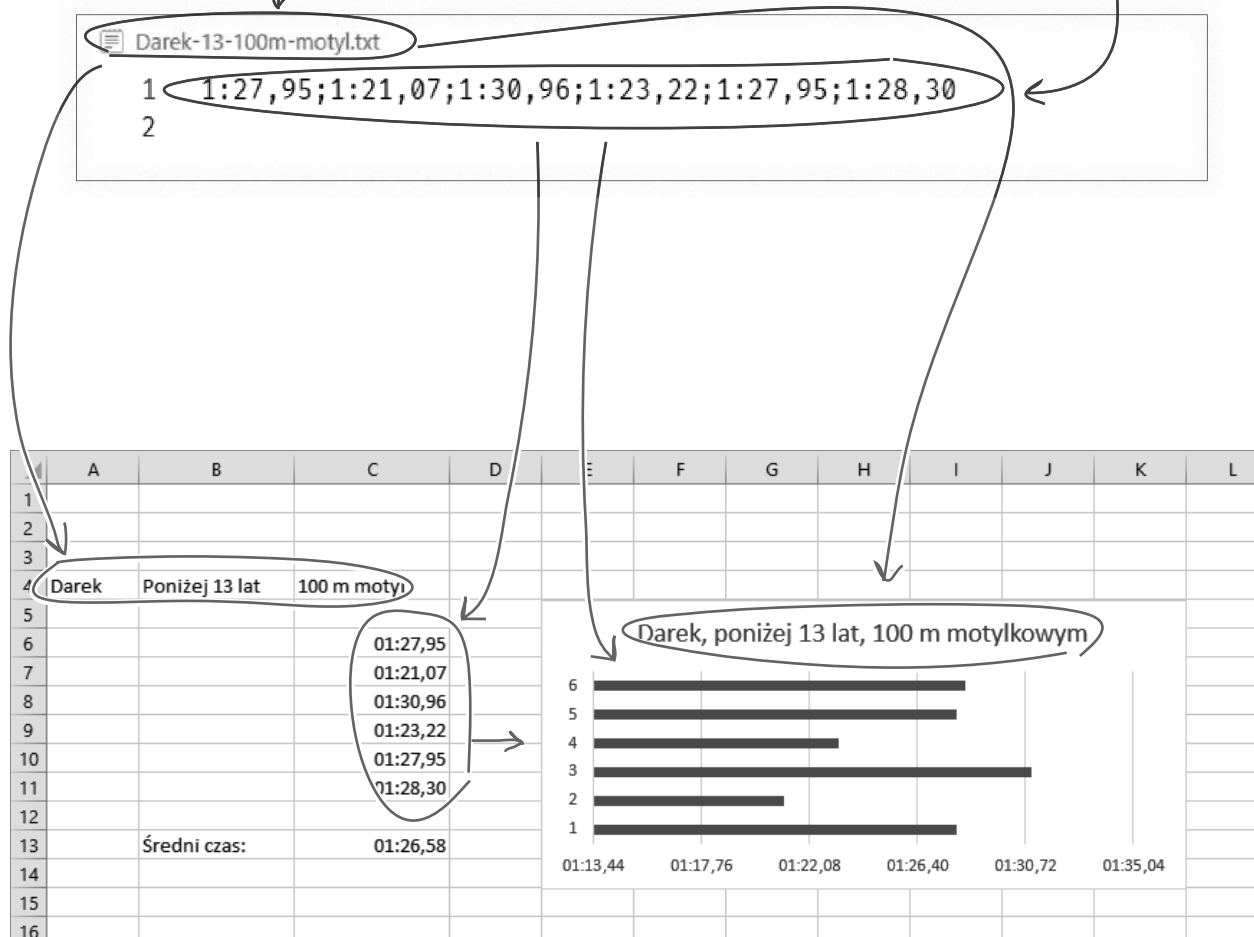
oraz obliczoną średnią (z zadania nr 2)

Plik i arkusz kalkulacyjny są ze sobą „powiązane”

Zanim przejdziemy do zadania 1, poświęć chwilę na sprawdzenie, w jaki sposób dane umieszczone w nazwie pliku oraz dane w tym pliku zapisane odnoszą się do arkusza kalkulacyjnego trenera:

Dane umieszczone w nazwie pliku pojawiają się dwukrotnie w arkuszu kalkulacyjnym (jako nagłówki).

Dane z treści pliku pojawiają się w arkuszu kalkulacyjnym dwukrotnie: raz w kolumnie (która obsługuje obliczanie średniej) i ponownie na wykresie słupkowym (gdzie wartości czasu odpowiadają długości słupka).



Nasze pierwsze zadanie: wyodrębnić dane z nazwy pliku

Na razie planujemy skoncentrować się na jednym pliku, a konkretnie na pliku zawierającym dane dotyczące czasów Darka z grupy wiekowej poniżej 13 lat na dystansie 100 m motylkiem.

Przypomnijmy, że plik zawierający potrzebne dane nosi nazwę *Darek-13-100m-motyl.txt*.

Utwórz zatem notatnik Jupytera o nazwie *Darek.ipynb* i zapisz go w swoim katalogu *Nauka*. Śledź postępy w swoim notatniku, abyśmy wspólnie realizowali zadanie nr 1.

Pamiętaj: Aby utworzyć nowy notatnik w VS Code, należy wybrać z menu głównego opcję *Plik*, potem *Nowy plik...*, a następnie opcję *Jupyter Notebook*.

Wpisz to w pierwszej komórce kodu.

Utwórz nową zmienną o nazwie „fn”, skrót od słów „file name” – „nazwa pliku”.

```
fn = "Darek-13-100m-motyl.txt"
```

Operator przypisania (=) zapisuje łańcuch znaków w zmiennej „fn”. Ten łańcuch zawiera nazwę pliku (utworzoną przez stoper).

Jest to łańcuch znaków (to, że jest on zapisany w cudzysłowach, to zapewne duża wskazówka, prawda?).

Nie zapomnij nacisnąć kombinacji klawiszy „Shift+Enter”, aby wykonać kod zapisany w komórce notatnika.

W Pythonie wszystko jest obiektem

Podobnie jak w innych obiektowych językach programowania, także w Pythonie obiekty mogą mieć *atrybuty* i/lub *metody*, które są z nimi powiązane.

Nie oznacza to, że aby używać Pythona, musisz być czarodziejem w programowaniu obiektowym, nic bardziej mylnego.

Oznacza to natomiast, że możesz przyjąć za pewnik, że bez względu na to, z czym pracujesz w swoim kodzie, *podwójne mambo* `print(dir(fn))` zwróci informacje o istniejących atrybutach i/lub metodach, jeśli takie istnieją. Ma to znaczenie, gdy eksperymentujesz z narzędziem REPL Pythona (co robisz w swoich notatnikach Jupytera). Często warto zadawać sobie proste pytanie: *Co znajduje się w obiekcie fn i co mogę z nim zrobić?*, na które odpowiedź można znaleźć, używając jednego wiersza kodu:

```
print(dir(fn))
```

„Podwójne mambo”



Możliwe, że nie będziesz w stanie powstrzymać się przed tym, żeby wstać i wykonać mały taniec radości. Idź na całość – nikt nie patrzy 😊

Łańcuch znaków to obiekt mający atrybuty



Wiesz już, jak wyświetlić listę atrybutów dowolnego obiektu za pomocą podwójnego mambo `print dir`, zobaczymy więc, co pojawi się po zapytaniu o zmienną `fn`:

Pamiętaj, aby to wywołanie czytać od środka na zewnątrz: przekazujemy nazwę interesującej nas zmiennej do wbudowanej funkcji „dir”, a następnie zwrócone przez nią wyniki przekazujemy do funkcji „print”.

```
print(dir(fn))
```

```
['_add_', '_class_', '_contains_', '_delattr_', '_dir_', '_doc_', '_eq_',
'_format_', '_ge_', '_getattr_', '_getitem_', '_getnewargs_', '_gt_', '_hash_',
'_init_', '_init_subclass_', '_iter_', '_le_', '_len_', '_lt_', '_mod_', '_mul_',
'_ne_', '_new_', '_reduce_', '_reduce_ex_', '_repr_', '_rmod_', '_rmul_',
'_setattr_', '_sizeof_', '_str_', '_subclasshook_', 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum',
'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

Z technicznego punktu widzenia użycie funkcji „print” nie jest konieczne, ale pozwala wyświetlić tę dużą listę atrybutów w poziomie (co jest nieco łatwiejsze do przetworzenia dla mózgu).

Jak wspomnieliśmy w rozdziale otwierającym, na razie możesz zignorować wszystkie te nazwy z podwójnymi podkreśleniami. Metody obiektu łańcucha znaków zaczynają się od „capitalize” i kończą na „zfill”. Jest ich dużo, prawda?

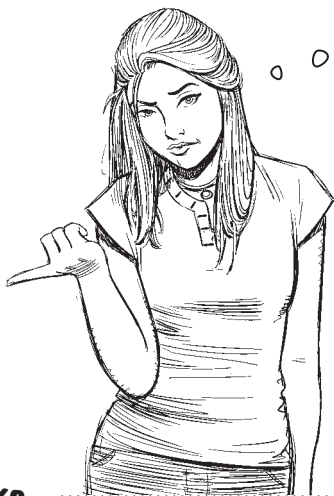
Daj sobie chwilkę, by docenić to, na co teraz patrzysz

Jeśli patrzysz na tę długą listę atrybutów i myślisz, że jest tu strasznie dużo do nauczenia się, to zastanów się nad tym: w łańcuchy znaków jest wbudowanych bardzo dużo funkcjonalności, dla których *nie musisz* pisać kodu.

Nie istnieją
grupy pytania

P: Zauważyłem, że wyświetlając łańcuchy znaków, Python używa apostrofów, podczas gdy w całym pokazanym kodzie stosowane są cudzysłowy. Dlaczego tak jest?

O: To, którego z tych zapisów użyjesz w swoim kodzie, zależy od osobistych preferencji, ale możesz zastosować dowolny z nich (a także mieszać je ze sobą). Po prostu my lubimy cudzysłowy.



Nawet jeśli zignoruję podwójne podkreślenia, i tak jest to naprawdę spora lista. Czy muszę zapamiętać ją całą?



Relaks

Wywołanie `print dir` wyświetliło dużą listę, ale wystarczy myśleć tylko o połowie jej zawartości.

Możesz bezpiecznie zignorować wszystkie metody, które zaczynają się i kończą dwoma znakami podkreślenia, takie jak `__add__` i `__ne__`. Są to „magiczne metody” tego obiektu i mają swoje przeznaczenie, ale jest zbyt wcześnie, abyś musiał się martwić o to, co one robią i jak można ich używać. Zamiast tego skoncentruj się na pozostałych metodach z tej listy.

Nie istnieją głupie pytania

P: Jeśli te metody z podwójnymi podkreśleniami nie są ważne, dlaczego znajdują się na liście zwracanej przez `dir`?

O: Nie chodzi o to, że nie są ważne, chodzi raczej o to, że na tym etapie nie musisz przejmować się tym, co one robią. Zaufaj nam — kiedy będziesz chciał zrozumieć, co robią metody z tymi podwójnymi podkreśleniami, powiemy ci. Słowo skauta!

P: Czy istnieje sposób, aby dowiedzieć się więcej o tym, co robi dana metoda?

O: Tak. Przypomnijmy, że w poprzednim rozdziale pokazaliśmy, jak korzystać z wbudowanej funkcji `help`. Więcej na ten temat dowiesz się za chwilę.

P: Trochę to wszystko skomplikowane, ten cały cyrk z podkreśleniami w nazwach, prawda?

O: Tak, to prawda. Dlatego hardcorowi programiści Pythona, używający angielskiego slangu, zamiast mówić „podwójne podkreślenia”, używają skrótu angielskiego odpowiednika tych słów („double underscore”) — „dunder”. I tak, zamiast „podwójne podkreślenie `add` podwójne podkreślenie” mówią „dunder `add`”. Jeśli usłyszysz, że ktoś mówi o metodzie „dunder `exit`”, to tak naprawdę będzie miał na myśli metodę `__exit__`. Ale hełło... pisanie o „dunderach” byłoby zbyt niepoważne nawet jak na TĘ KSIĄŻKĘ!

Zaostż ołówek



Spróbuj wykonać dwie metody łańcuchów znaków. Każdy z wierszy kodu pokazanych poniżej wprowadź do nowej, pustej komórki kodu w notatniku. Wykonaj każdą z nich, a następnie zanotuj (w odpowiednim miejscu), co według Ciebie robi każda z tych funkcji.

`fn.upper()` _____

`fn.lower()` _____

→ Odpowiedź znajdziesz na stronie 54.

Zanim przyjrę się odpowiedziom, czy mógłbyś odpowiedzieć na krótkie pytanie? O co chodzi z tą kropką powyżej? Wydaje mi się to ważne...



Nie ma problemu. Przy okazji, świetne pytanie.

Ten znak to operator **kropki** Pythona, który daje dostęp do obiektu. Kiedy używasz go z nazwą metody (wraz z nawiasami), „mówisz” Pythonowi, aby *wywołał* wskazaną metodę. Różni się to nieco od wbudowanych funkcji, które są wywoływane jak... funkcje. Na przykład `fn.len()` zwraca rozmiar obiektu, do którego odnosi się zmienna `fn`.

Próba wywołania `fn.len()` byłaby błędem, ponieważ nie ma takiej metody, tak samo jak błędem byłoby wypróbowanie `upper(fn)`, ponieważ taka funkcja wbudowana nie istnieje.

Pomyśl o tym w ten sposób: metody są specyficzne dla obiektów, podczas gdy wbudowane funkcje dostarczają *ogólnych* funkcjonalności, które można zastosować do obiektów większości typów.

Zaostrz ołówek

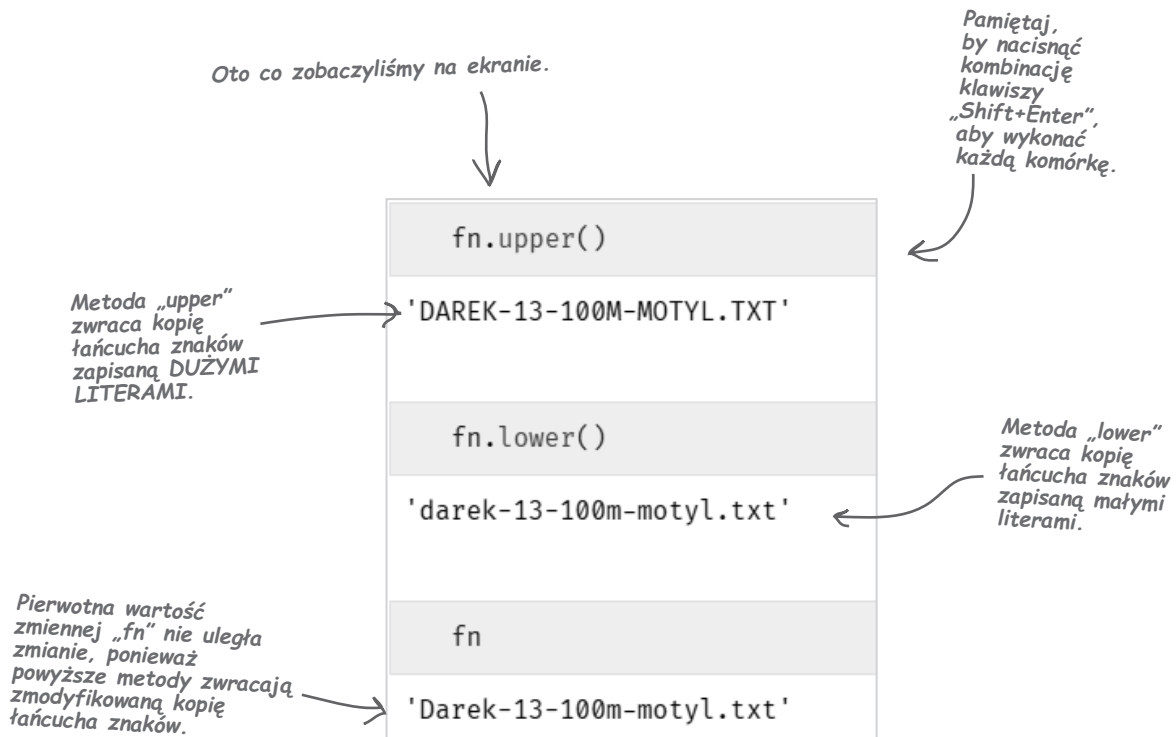
Rozwiązanie ze strony 53.

Poproszono Cię o uruchomienie dwóch podanych metod. Każdy z przedstawionych poniżej wierszy kodu miałeś wpisać do nowej, pustej komórki notatnika. Następnie miałeś wykonać każdą z nich i zanotować (w wyznaczonym miejscu), co według Ciebie zrobiła każda z metod. Oto co naszym zdaniem tutaj się dzieje:

fn.upper() Zwraca kopię wartości, do której odwołuje się zmienna „fn”
zapisaną DUŻYMI literami.

fn.lower() Zwraca kopię wartości, do której odwołuje się zmienna „fn”
zapisaną małymi literami.

Oto co zobaczyliśmy na ekranie.



Więc... metody „upper” i „lower” zwracają zmodyfikowane kopie na podstawie wartości zmiennej „fn”?



Tak, to prawda.

Wartości zwracane przez metody upper i lower są *nowymi* obiektami łańcucha znaków, z których jeden jest zapisany **DUŻYMI LITERAMI**, a drugi *małymi*. Z dotychczasową wartością zmiennej fn nic się nie dzieje, co potwierdzają wyniki wyświetlone na dole poprzedniej strony.

Jest to zgodne z projektem, ponieważ wszystkie łańcuchy znaków w Pythonie są *niezmiennie* (nie mogą się zmieniać). To pomaga wyjaśnić, dlaczego metody upper i lower zwracają zmodyfikowane kopie łańcucha znaków zapisanego w zmiennej fn — po prostu łańcuch fn jest niezmienny.

Nie istnieją głupie pytania

P: Czy jest łatwy sposób na stwierdzenie, czy zmienna jest niezmienna, czy jej wartość można modyfikować?

O: Cóż... zasada jest taka: liczby, łańcuchy znaków, wartości logiczne i krotki są niezmiennie, natomiast większość innych elementów jest modyfikowalna (dotyczy to na przykład list, zbiorów i słowników). Sprawy mogą się nieco skomplikować, jeśli spróbujesz określić to w czasie wykonywania kodu. Jedną z możliwych technik jest przekazanie zmiennej do wbudowanej funkcji hash. Jeśli przekazana zmienna jest niezmienna, funkcja hash zwróci liczbę. Jeśli przekazana zmienna jest modyfikowalna, wykonanie tej funkcji nie powiedzie się — zakończy się błędem TypeError, który należałoby obsłużyć, używając jakiegoś kodu przechwytyującego wyjątki. Ale jak na razie jest chyba zbyt wcześnie by o tym mówić...

Pobieramy dane pływaka z nazwy pliku

Przypomnij sobie trzy podzadania zidentyfikowane wcześniej dla zadania nr 1:

- 1a. Pobrać nazwę pliku. ✓
- 1b. Podzielić nazwę pliku w miejscach wystąpienia znaku „-”.
- 1c. Zapisać imię pływaka, jego grupę wiekową, dystans oraz styl w zmiennych (by można ich było używać w dalszej części kodu).

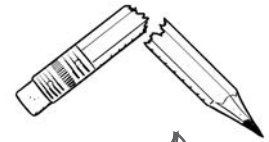
Ponieważ nazwa pliku jest już zapisana w zmiennej `fn`, przyjmijmy, że podzadanie (a) zostało wykonane. (W kolejnym rozdziale zajmiemy się uzyskaniem wszystkich nazw plików od systemu operacyjnego).

Podzadanie (b) polega na podzieleniu nazwy pliku w miejscach występowania znaku „-” i zapewne słusznie przypuszczasz, że może nam w nim pomóc jedna z metod łańcuchów znaków. Ale która? Jest ich aż 47!

Usunęliśmy magiczne metody (czyli te, których nazwy są zapisane pomiędzy znakami podkreślenia) z listy metod zwracanych przez „`print dir`” dla łańcucha znaków „`fn`”.

```
'capitalize', 'casefold', 'center', 'count', 'encode',  
'endswith', 'expandtabs', 'find', 'format', 'format_map',  
'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal',  
'isdigit', 'isidentifier', 'islower', 'isnumeric',  
'isprintable', 'isspace', 'istitle', 'isupper', 'join',  
'ljust', 'lower', 'lstrip', 'maketrans', 'partition',  
'removeprefix', 'removesuffix', 'replace', 'rfind',  
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',  
'split', 'splitlines', 'startswith', 'strip', 'swapcase',  
'title', 'translate', 'upper', 'zfill'
```

Jest to długa lista metod łańcuchów znaków. Choć łatwo zgadnąć, co robią metody „`upper`” i „`lower`”, to jednak w przypadku niektórych innych metod wcale nie jest to oczywiste; dotyczy to na przykład takich metod jak: „`casefold`”, „`format_map`” czy „`zfill`”. Tym, czego szukasz, jest metoda, która mogłaby nam pomóc w rozwiązaniu podzadania (b).



Cóż... to też można by nazwać „podziałem”, ale uwierz nam, nie o niego nam chodziło 😊

Na tej liście nie ma ani metody „`break`”, ani „`break apart`”, ale jest metoda o nazwie `split`, która brzmi zachęcająco. Dowiedzmy się, co ona robi.

Nie próbuj zgadywać, co robi dana metoda...

Przeczytaj dokumentację metody!

Oczywiście większość programistów wolałaby zjeść szklanke niż szukać i czytać dokumentację, ponieważ twierdzą oni, że życie jest zbyt krótkie, zwłaszcza gdy trzeba pisać kod. Zazwyczaj największą irytację związaną z korzystaniem z dokumentacji wywołuje poszukiwanie potrzebnych informacji. Dlatego Python ułatwia znajdowanie i wyświetlanie odpowiedniej dokumentacji — służy do tego wbudowana funkcja `help`.

Niestety Python nie może czytać dokumentacji za Ciebie, więc nadal będziesz musiał zrobić to sam. Ale funkcja `help` pozwala uniknąć zmiany kontekstu, wychodzenia z VS Code, otwierania przeglądarki internetowej, a następnie przeszukiwania dokumentacji.

Aby wyświetlić dokumentację dowolnej metody, użyj funkcji `help`, jak poniżej:



Przekazujemy nazwę metody do funkcji „help”. Pamiętaj, aby odnosić się do metody w kontekście zmiennej, do której ona należy, używając notacji z kropką. Ważne jest, aby przy tym pamiętać, że nie wywołujemy metody, tylko przekazujemy do funkcji „help” jej nazwę.

To jest sygnatura metody, która określa sposób jej wywołania.

```
help(fn.split)
```

Help on built-in function split:

```
split(sep=None, maxsplit=-1) method of builtins.str instance
Return a list of the words in the string, using sep as the delimiter string.
```

```
sep
```

The delimiter according which to split the string.

None (the default value) means split according to any whitespace, and discard empty strings from the result.

```
maxsplit
```

Maximum number of splits to do.

-1 (the default value) means no limit.

*Ten wiersz zawiera krótki opis działania metody, więc dla nas to właśnie on jest *najważniejszy*.*

Ta część dokumentacji opisuje znaczenie parametrów. Uwaga: oba parametry mają wartości domyślne, które zostaną użyte, gdy którykolwiek z dwóch argumentów nie zostanie podany w wywołaniu.

Na podstawie szybkiej lektury tej dokumentacji wydaje się, że metoda `split` jest tym, czego potrzebujemy. Przyjrzyjmy się zatem bliżej tej metodzie w działaniu.

Dzielenie łańcuchów znaków na części

Dokumentacja metody `split` wyraźnie sugeruje jej wbudowaną moc, ale bardzo wiele można osiągnąć za pomocą kilku prostych przykładów. Domyślnie `split` używa tak zwanych *białych znaków* (ang. *whitespaces*) jako separatora. Rozważmy przykładowy łańcuch znaków, który jest zapisany w zmiennej `title`:

W Pythonie pod pojęciem „białych znaków” rozumiane są: odstęp (spacja), tabulator, znak nowego wiersza, znak powrotu karetki, znak wysunięcia strony (formfeed) i pionowy tabulator.

Nie ma tu nic szczególnie interesującego. W zmiennej „title” zapisujemy łańcuch znaków.

```
title = "Nareszcie, wielkie dzięki za nasze ryby."
```

Ponieważ odstęp jest białym znakiem, poniższe wywołanie dzieli powyższy łańcuch znaków w przedstawiony sposób, tworząc listę łańcuchów z poszczególnymi słowami:

```
title.split()
```

Nie podano żadnych argumentów, więc wykorzystany zostanie domyślny sposób działania metody „split”: dzielenie łańcucha w miejscach występowania białych znaków.

```
['Nareszcie,', 'wielkie', 'dzięki', 'za', 'nasze', 'ryby.']
```

Nawiasy kwadratowe, otaczające zwrócone wyniki, są pierwszą wskazówką, że mamy do czynienia z listą.

Dane oddzielone przecinkami to druga wskazówka, że zwrócona została lista. W tym przykładzie wszystkie wartości zapisane na tej liście są łańcuchami znaków.

Domyślny separator można łatwo zmienić, podając łańcuch znaków jako pierwszy argument wywołania metody `split`:

```
title.split(", ")
```

Domyślny separator jest zastępowany separatorem utworzonym z kombinacji znaków przecinka i odstępu.

```
['Nareszcie,', 'wielkie dzięki za nasze ryby.']
```

To wywołanie metody „split” zwraca inną listę, jednak tym razem nie widać na niej „słów”, a raczej dłuższe fragmenty początkowego łańcucha (co jest całkowicie w porządku).

Uwaga: ponieważ wszystkie łańcuchy znaków są niezmiennie, pierwotna zawartość zmiennej „title” *nie* uległa zmianie.



Jazda próbna

Wróćmy do naszego notatnika *Darek.ipynb* i spróbujmy zbadać, co się stanie, gdy użyjemy metody `split` przeciwko zmiennej `fn`. Jak zawsze zapraszamy, byś przekonał się o tym, wykonując to u siebie na komputerze.

Na początek zobaczmy, co się stanie, gdy wywołamy metodę `split` bez podawania żadnych argumentów:

Używając operatora kropki, wywołujemy metodę „`split`” na rzecz zmiennej „`fn`” (bez przekazywania jakichkolwiek argumentów).

```
fn.split()
```

```
['Darek-13-100m-motyl.txt']
```

A niech to! To nie spowodowało nic poza umieszczeniem łańcucha pomiędzy nawiasami kwadratowymi. Nie jest to zbyt użyteczne, prawda?

Ponieważ metoda `split` nie znalazła białych znaków w łańcuchu `fn`, zwróciła ten łańcuch w całości jako pierwszy i jedyny element listy wynikowej. To nie jest to, o co nam chodziło, więc spróbujmy ponownie, tym razem używając jako separatora znaku „-”:

Przekazujemy znak „-” jako argument wywołania metody „`split`”.

```
fn.split("-")
```

```
['Darek', '13', '100m', 'motyl.txt']
```

To wygląda znacznie lepiej. Zamiast jednego łańcucha mamy teraz cztery mniejsze. Pierwotny łańcuch został podzielony w miejscach występowania znaku „-”.

Hura! Spójrz na to!
Ale... czy tam na końcu powinno być to „.txt”?



Emmm... to nie wygląda do końca dobrze, prawda?

Jesteśmy prawie na miejscu, ale wciąż mamy coś do zrobienia.

Wciąż zostaje nam coś do zrobienia

Kusi, by spojrzeć na listę podzadań, wziąć długopis i postawić śliczną, satysfakcjonującą parafkę przy podzadaniu (b), prawda?

- 1a Pobrać nazwę pliku. ✓
- 1b Podzielić nazwę pliku w miejscach wystąpienia znaku „-”.
- 1c Zapisać imię pływaka, jego grupę wiekową, dystans oraz styl w zmiennych (by można ich było używać w dalszej części kodu).

Byłoby to jednak *przedwczesne*. Jak zauważył trener, mamy dodatkowe dane, które nie są potrzebne, więc musimy przyjrzeć się bliżej liście utworzonej przez wywołanie metody `split`:

Na pierwszy rzut oka wyniki są okej, ponieważ pierwotny łańcuch znaków został podzielony w miejscu występowania znaków minusa.

```
fn.split("-")
```

```
['Darek', '13', '100m', 'motyl.txt']
```

Ale jako część ostatniego „słowa” zostało dołączone rozszerzenie pliku („.txt”). Tak naprawdę nie jest to część nazwy żadnego stylu pływania, więc należy je usunąć, prawda?



MOC UMYSŁU

Czy istnieje lepsza strategia niż dzielenie wyjściowego łańcucha w miejscach występowania znaku „-”?

Zaostrz ołówek



Poświęćmy chwilę na utrwalenie zrozumienia sposobu działania metody `split`.
Oto jeszcze raz instrukcja przypisania zapisująca łańcuch znaków w zmiennej `fn`:

```
fn = "Darek-13-100m-motył.txt"
```

Bez wcześniejszego wykonywania przedstawionych instrukcji w swoim notatniku sprawdź, czy potrafisz opisać, co robi każda z nich, i zapisz swoje odpowiedzi we właściwych miejscach. Jeśli utkniesz, nie martw się: nasze odpowiedzi znajdują się na następnej stronie. A tak przy okazji, kiedy już spróbujesz wyobrazić sobie, co robi każda instrukcja, możesz sprawdzić swoją pracę w VS Code (tylko nie zaczynaj od tego!).

1

```
fn.split("13")
```

2

```
fn.split("-1")
```

3

```
fn.split(".")
```

4

```
fn.split("-.")
```

5

```
fn.split("-").split(".")
```

→ Odpowiedź znajdziesz na stronie 62.

Zaostrz ołówki

Rozwiązanie ze strony 61.

Miałeś poświęcić chwilę na to, aby utrwalić sobie sposób działania metody `split`.

Bez wcześniejszego uruchamiania przedstawionych instrukcji w swoim notatniku miałeś sprawdzić, czy potrafisz wyobrazić sobie, co robi każda z nich, a następnie zapisać odpowiedzi w podanych miejscach. Nasze odpowiedzi znajdują się na tej i następnej stronie, wraz z wynikami wyświetlonymi w VS Code po wykonaniu każdej z instrukcji. W jakim stopniu Twoje odpowiedzi odpowiadają naszym?

1

```
fn.split("13")
```

Dzieli łańcuch znaków w miejscu występowania łańcucha „13”.

Czyli w tym miejscu, gdzie znaki „1” i „3” występują razem.

Ponieważ „13” pojawia się tylko raz, wyjściowy łańcuch znaków jest dzielony na dwie części.

```
→ ['Darek-', '-100m-motył.txt']
```

↑
Fragment przed „13”.

↑
Fragment po „13”.

Ten przykład pokazuje, że separatorem przekazywanym do wywołania metody „split” może być łańcuch znaków o dowolnej długości.

2

```
fn.split("-1")
```

Dzieli łańcuch znaków w miejscu występowania łańcucha „-1”.

Czyli w miejscu, gdzie znaki „-” i „1” występują razem.

Tym razem łańcuch wyjściowy jest dzielony na trzy części, ponieważ fragment „-1” pojawia się w nim dwukrotnie.

```
→ ['Darek', '3', '00m-motył.txt']
```

Zwróć uwagę, że metoda „split” usuwa separator ze zwracanych wyników.

Podobnie jak we wszystkich wywołaniach metody „split” wynikiem jest lista (zwróć uwagę na nawiasy kwadratowe).

3

`fn.split(".")`Dzieli łańcuch znaków w miejscach występowania znaku „.”,czyli kropki.

Nasz łańcuch znaków zawiera jedną kropkę, więc utworzona lista będzie zawierać dwa obiekty utworzone z fragmentów łańcucha przed znakiem kropki i po nim.

→ `['Darek-13-100m-motyl', 'txt']`

4

`fn.split("-.")`Dzieli łańcuch znaków w miejscach występowania łańcucha „-.”,czyli w miejscach, gdzie znaki „-” i „.” występują oboksiebie.

← `['Darek-13-100m-motyl.txt']`

Chociaż znaki „-” i „.” pojawiają się w łańcuchu, nie występują razem, więc nie ma podziału. Nie jest to błąd. Metoda „split” zwraca początkowy łańcuch znaków jako jedyny element wynikowej listy (zgodnie ze swoim domyślnym działaniem).

5

`fn.split("-").split(".")`Dzieli łańcuch znaków w miejscach występowania znaku „-”,a następnie wykonuje kolejny podział z użyciem jakoseparatora znaku „.” (czyli kropki)...

O kurczę!



← Nie. To działa inaczej!

AttributeError

Traceback (most recent call last)

Cell In [14], line 1

----> 1 `fn.split("-").split(".")`

AttributeError: 'list' object has no attribute 'split'

↖ W ostatnim przykładzie wystąpił błąd, który spowodował wyświetlenie włochatego, przerażającego komunikatu o błędzie. Przewróć kartkę, aby dowiedzieć się więcej o tym, co się tutaj stało.

Komunikaty błędów czytaj od dołu do góry

Ostatni przykład w ostatnim ćwiczeniu wygenerował komunikat o błędzie czasu wykonywania, który prawdopodobnie sprawił, że zaczęłeś drapać się po głowie:

AttributeError Traceback (most recent call last)

Cell In [14], line 1

----> 1 fn.split("-").split(".")

AttributeError: 'list' object has no attribute 'split'

← Strzałka wskazuje, w którym miejscu kodu wystąpił błąd.

↑
Sztuczka pozwalająca zrozumieć komunikaty o błędach Pythona polega na czytaniu ich od dołu do góry.

←
To jest najważniejszy wiersz, więc zawsze czytaj go jako pierwszy, ponieważ to tutaj dowiesz się, CO poszło nie tak. Reszta komunikatu o błędzie mówi, GDZIE coś poszło nie tak.

Powiedziano mi, że jeśli to wypiję, zyskam panowanie nad supermocami Pythona. A więc zaczynamy... DNEM DO GÓRY I PIJEMY!



Ech... Dobra.

Jak tam sobie chcesz.

Po prostu pamiętaj, aby komunikaty o błędach Pythona zawsze czytać od dołu do góry, a wszystko będzie dobrze (niezależnie od magicznych mikstur). Zauważ również, że Python nazywa swoje komunikaty o błędach **śladami wykonania** (ang. *traceback*).

Teraz... co ten konkretny „ślad” próbuje Ci powiedzieć?

Uważaj na łączenie wywołań metod

Prezentując ten ostatni przykład, mieliśmy ściśle określony zamiar: chodziło nam o utworzenie sekwencji wywołań metody `split`, z których pierwsze dzieliłoby początkowy łańcuch znaków w miejscach występowania znaku „-”, a drugie — w miejscach występowania znaku „.”:

```
fn.split("-").split(".")
```

Zgodnie z technikami Pythona łączenie wywołań metod w ten sposób jest dozwolone, ale należy zachować ostrożność.

Oczywiście ten wiersz kodu nie zadziałał zgodnie z naszymi oczekiwaniami, co jest kłopotliwe, ponieważ pomysł był rozsądny: chcieliśmy dwukrotnie podzielić łańcuch znaków, aby oddzielić fragmenty "motyl" i "txt". Ale spójrz na wyświetlony komunikat o błędzie:

```
AttributeError: 'list' object has no attribute 'split'
```

Ale o co chodzi!?! Python narzeka, że próbujemy zrobić coś z listą, ale przecież wiemy, że zmienna „fn” zawiera łańcuch znaków. Co się dzieje?

Myślę, że wiem, co się tutaj dzieje. Pierwsze wywołanie dzieli początkowy łańcuch znaków w miejscach występowania „-”, tworząc listę, na rzecz której ponownie próbujemy wywołać metodę „split”. Przecież metoda ta może być wywoływana na rzecz łańcuchów znaków, a tu zamiast łańcucha pojawia się lista. Prawda?

Tak, dokładnie to się dzieje.

Pierwsze wywołanie `split` działa poprawnie, dzieląc obiekt łańcucha znaków w miejscu występowania minusów („-”) i tworzy listę. Lista ta jest następnie używana do wywołania kolejnej metody w łańcuchu, którą również jest `split`. Problem polega na tym, że listy nie mają metody `split`, więc próba wywołania `split` na rzecz listy nie ma sensu... i w efekcie Python poddaje się, zgłaszając błąd `AttributeError`.

A teraz, skoro już wiesz, co jest grane, czy wiesz też, jak to poprawić?



Rozmowa w boksie

Olek: Myślę, że powinniśmy wstrzymać prace nad kodem dla trenera i najpierw dowiedzieć się wszystkiego o listach...

Jan: Nie jestem tego taki pewien. Choć listy są ważną strukturą danych Pythona, nie mam pewności, czy to jest problem z listą, niezależnie od tego, co twierdzi ślad wykonania.

Marta: Zgadzam się. Próbujemy manipulować *łańcuchem* `fn`, a nie listą. Fakt, że ślad wykonania odwołuje się do listy, jest trochę mylący, a dzieje się tak, ponieważ metoda `split` zwraca listę.

Olek: A nie możemy wywołać `split` na liście, prawda?

Jan: Nie, nie możemy, ponieważ listy, w odróżnieniu od łańcuchów znaków, nie udostępniają wbudowanej metody `split`. I właśnie to mówi nam ślad wykonania: w listach nie ma czegoś takiego jak metoda `split`.

Marta: Gdybyśmy wiedzieli nieco więcej o listach, moglibyśmy pobrać fragment `"motyl.txt"` i podzielić go ponownie, prawda?

Jan: Tak, ale nie jestem pewien, czy musimy to robić. Myślę, że może istnieć inna metoda łańcuchów znaków, która mogłaby nam pomóc.

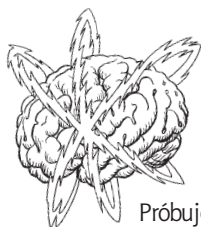
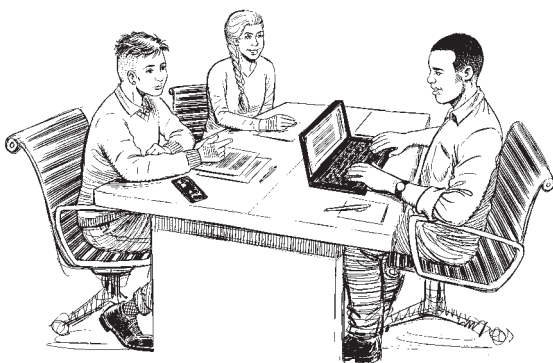
Olek: Wygląda na to, że nie mamy szczęścia?

Marta: Nie. Musimy tylko wszystko przemyśleć...

Jan: Pamiętaj, że `fn` jest łańcuchem. Zaczynj więc od zadania sobie pytania, czy są jakieś wbudowane metody łańcuchów, które mogłyby nam pomóc.

Olek: Jak mogę to zrobić ponownie?

Jan: Poproś o pomoc starego przyjaciela: podwójne mambo `print dir` — użyj go, aby wyświetlić wszystkie wbudowane metody wszystkich łańcuchów znaków.



MOC UMYSŁU

Próbujesz pozbyć się tego `".txt"` na końcu początkowego łańcucha znaków. Oto lista metod udostępnianych przez obiekty łańcuchów znaków. Czy któraś z tych nazw metod rzuca się w oczy?

```
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',  
'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit',  
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',  
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix',  
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',  
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill'
```

Wypróbujmy inną metodę łańcuchów znaków

Przejrzymy listę metod łańcuchów znaków, aby sprawdzić, czy coś nie rzuca się nam w oczy:

```
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit',
'identifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill'
```

Ta ma interesującą nazwę.

Podobnie jak w przypadku każdej innej metody, użyj wbudowanej funkcji help, by wyświetlić szczegółowe informacje na temat działania metody removesuffix:

```
help(fn.removesuffix)
```

Help on built-in function removesuffix:

removesuffix(suffix, /) method of builtins.str instance

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[: -len(suffix)]. Otherwise, return a copy of the original string.

To brzmi jak coś, o co nam chodzi.

Ta część dokumentacji metody „removesuffix” to jakiś bełkot, zwłaszcza ten fragment „[: -len(suffix)]”, prawda? Na razie nie przejmuj się tym akapitem. W dalszej części książki dowiesz się, jak działa ten zapis (kiedy, miejmy nadzieję, ta dokumentacja będzie już miała sens). Zresztą prawdopodobnie przestałeś czytać już po pierwszym wierszu, który mówi wszystko, co musisz wiedzieć 😊

Wypróbuj tę metodę w swoim notatniku.



Jazda próbna

Wróćmy do notatnika *Darek.ipynb* i dodajmy do niego kilka testowych wywołań metody `removesuffix`, aby sprawdzić, czy może ona pomóc w pozbyciu się niechcianego rozszerzenia `".txt"`:



Teraz, kiedy już dysponujesz metodą, która robi to, czego potrzebujesz, możesz połączyć wywołanie `removesuffix` z wywołaniem `split`, aby wyodrębnić potrzebne dane z łańcucha zapisanego w zmiennej `fn`:

Wywołujemy metodę „`removesuffix`” na początkowym łańcuchu znaków, a następnie, po usunięciu fragmentu `".txt"`, wywołujemy „`split`”, aby podzielić to, co pozostało w miejscach występowania znaku `"-"`, i w efekcie utworzyć listę zawierającą potrzebne nam dane.

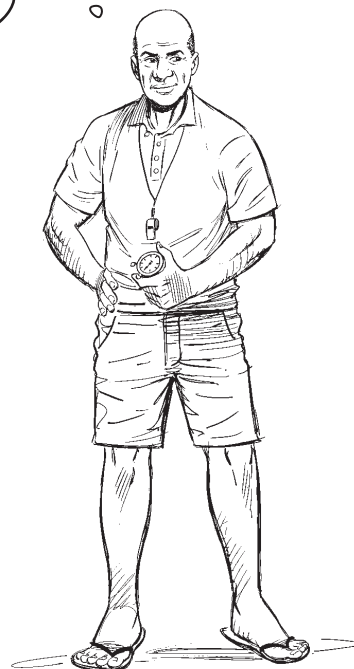
```
fn.removesuffix(".txt").split("-")
```

```
['Darek', '13', '100m', 'motyl']
```

Zapowiada się nieźle!

Jeśli wykonujesz te wszystkie przykłady w swoim notatniku, możesz w tym momencie zapisać swoją pracę. Notatniki możesz zapisywać tak często, jak chcesz (tak jak każdy inny edytowany plik).

Dokładnie tak wyglądają dane na ekranie mojego inteligentnego stopera. Imię, grupa wiekowa, dystans i styl. To wielki postęp.



Dzięki!

Zostało nam już tylko jedno podzadanie, a potem zakończymy prace przewidziane w tym rozdziale.

1a Pobrać nazwę pliku. ✓

Nadszedł czas na postawienie parafki.

1b Podzielić nazwę pliku w miejscach wystąpienia znaku „-”. ✓

Nie wiemy jak Ty, ale my poczuliśmy się świetnie!

1c Zapisać imię pływaka, jego grupę wiekową, dystans oraz styl w zmiennych (by można ich było używać w dalszej części kodu).

↑
Ostatnia część zadania.

Pozostało nam jedynie utworzenie paru zmiennych

Ostatni wiersz kodu wygenerował listę z czterema wartościami, których potrzebuje trener, ale jak przypisać każdą z tych czterech wartości do odpowiedniej zmiennej?

Kombinacja wywołań „removesuffix” i „split”

```
fn.removesuffix(".txt").split("-")
```

```
['Darek', '13', '100m', 'motyl']
```

To jest lista (zwróć uwagę na nawiasy kwadratowe i przecinki oddzielające poszczególne wartości).

W tym momencie na pewno kusi Cię, aby zrobić sobie przerwę i dowiedzieć się trochę więcej o listach, co pozwoliłoby Ci manipulować powyższymi danymi i wybrać poszczególne wartości przed przypisaniem ich do odpowiednich zmiennych. Z pewnością mógłbyś tak postąpić, ale ponieważ tego typu operacje są bardzo powszechnym wymogiem, dlatego Python udostępnia wbudowaną technikę, która ułatwia to zadanie (i nie wymaga od Ciebie żadnej dodatkowej wiedzy na temat list).

Wielokrotne przypisywanie (określane także jako „rozpakowywanie”)

Chociaż ta koncepcja nie jest unikatowa dla Pythona, pojęcie *wielokrotnego przypisania* jest potężną możliwością tego języka.

Możliwość ta, znana również (w świecie Pythona) jako **rozpakowywanie**, pozwala na przypisanie do więcej niż jednej zmiennej umieszczonej po lewej stronie operatora przypisania odpowiedniej liczby wartości zapisanych po prawej stronie operatora.

Zróbmy krótką przerwę, aby dowiedzieć się, jak działa wielokrotne przypisywanie.

Wielokrotne przypisywanie... określane także jako „rozpakowywanie”... albo jako: magia.



Wielokrotne przypisywania pod lupą

Zachęcamy, byś wpisał prezentowane tu przykłady do swojego bieżącego notatnika, aby potwierdzić prawidłowość ich działania (w końcu są one wyświetlane w sposób charakterystyczny dla notatnika). Przykłady zaczynają się od instrukcji przypisania pojedynczej wartości, a następnie pokazują przypisanie wielokrotne, w którym dwie wartości po prawej stronie są przypisywane do odpowiednich zmiennych po lewej stronie:

Pojedynczej nazwie zmiennej po lewej stronie operatora przypisywana jest pojedyncza wartość zapisana po prawej stronie operatora.

W tym przypadku liczba 3,14 jest zapisywana w zmiennej „pie”.

```
pie = 3.14
```

```
pie
```

```
3.14
```

Możliwe jest również umieszczenie więcej niż jednej zmiennej po lewej stronie operatora przypisania oraz odpowiedniej liczby wartości po prawej stronie operatora. W tym przypadku mamy dwie zmienne.

```
pie, meaning = 3.14, 42
```

```
pie
```

```
3.14
```

```
meaning
```

```
42
```

Kolejność zmiennych po lewej stronie operatora przypisania jest dopasowywana do wartości po prawej stronie.



Mniam... zbliżamy się do końca tego rozdziału i wkrótce nadejdzie czas na przerwę. Być może należałoby zjeść kawałek ciasta... chociaż ostrzegamy przed dążeniem do 42 kawałków. Chociaż, po namyśle, doszliśmy do wniosku, że prawdopodobnie udałoby nam się osiągnąć 3,14...

Zwróć uwagę, że można dopasować dowolną liczbę zmiennych do wartości (o ile liczba zmiennych oraz liczba wartości zapisanych po obu stronach operatora przypisania będą równe). Python traktuje wartości po prawej stronie tak, jakby były listą, ale (w tym konkretnym przypadku) nie wymaga umieszczenia tej listy w nawiasach kwadratowych.



A zatem jeśli podam listę po prawej stronie operatora przypisania, to mogę przypisać wartości z listy do równoważnej liczby zmiennych po lewej stronie?

Tak, dokładnie.

Upewnij się tylko, że liczba wartości po prawej stronie odpowiada liczbie zmiennych po lewej stronie.

Programiści Pythona opisują listę jako „rozpakowaną” przed przypisaniem, co jest ich sposobem na wyrażenie, że wartości z listy są pobierane jedna po drugiej i przypisywane do kolejnych zmiennych. Pojedyncza lista jest *rozpakowywana*, a jej wartości są *przypisywane* do nazw zmiennych, jedna po drugiej.

Tak na marginesie: to samo dotyczy krotek (ale więcej na ten temat powiemy w kolejnym rozdziale).

Zaostrz ołówek



Weź ołówek i sprawdź, czy potrafisz wypełnić poniższe puste pola. Na podstawie tego, co już wiesz o wielokrotnym przypisywaniu (rozpakowywaniu), podaj poszczególne wiersze kodu, które przypisują prawidłowe rozpakowane wartości poszczególnym zmiennym. Podaj również wyświetlane dane wynikowe.

Tutaj brakuje jednego wiersza kodu. Dodaj go w wyznaczonym miejscu.

```
fn = "Darek-13-100m-motyl.txt"
```

```
print(swimmer)
print(age)
print(distance)
print(stroke)
```

W tych wierszach zapisz cztery wartości, które powinny zostać wyświetlone na ekranie po wykonaniu powyższej komórki notatnika.

Tu brakuje jednego wiersza kodu. Dodaj go w wyznaczonym miejscu.

```
fn = "Ania-10-50m-grzbietowy.txt"
```

```
print(swimmer)
print(age)
print(distance)
print(stroke)
```

Zapisz cztery wartości, które powinny zostać wyświetlone na ekranie po wykonaniu powyższej komórki notatnika.

→ Odpowiedź znajdziesz na stronie 74.



Zaostrz ołówki

Rozwiązanie ze strony 73.

Miałeś wziąć ołówki i sprawdzić, czy potrafisz wypełnić puste pola. Na podstawie swojej wiedzy na temat wielokrotnego przypisywania (rozpakowywania) miałeś podać wiersze kodu, które przypisują prawidłowe rozpakowane wartości do poszczególnych zmiennych. Miałeś również podać dane wynikowe, które zostaną wyświetlone. Poniżej znajduje się nasz kod. Czy Twój jest taki sam?

```
fn = "Darek-13-100m-motyl.txt"

swimmer, age, distance, stroke = fn.removesuffix(".txt").split("-")

print(swimmer)
print(age)
print(distance)
print(stroke)
```

Lista utworzona przez wywołanie „split” jest rozpakowywana i używana do przypisywania wartości wielu zmiennym.

Darek
13
100m
motyl

← To wygląda dobrze! Każdej zmiennej przypisano poprawną wartość wyodrębnioną z nazwy pliku.

```
fn = "Ania-10-50m-grzbietowy.txt"
swimmer, age, distance, stroke = fn.removesuffix(".txt").split("-")

print(swimmer)
print(age)
print(distance)
print(stroke)
```

← Nazwa pliku została zmieniona, ale nie kod.

Ania
10
50m
grzbietowy

← Zgodnie z oczekiwaniami, ze względu na użycie innej nazwy pliku, widoczne są różne wartości wynikowe.

Zadanie nr 1 zostało wykonane!

Przypomnijmy sobie listę czynności składających się na to zadanie:

- 1a Pobrać nazwę pliku. ✓
- 1b Podzielić nazwę pliku w miejscach wystąpienia znaku „-”. ✓
- 1c Zapisać imię pływaka, jego grupę wiekową, dystans oraz styl w zmiennych (by można ich było używać w dalszej części kodu). ✓

Zadanie nr 1:
wyodrębnienie
danych
z nazwy pliku

Gdy zmienna `fn` zawiera nazwę pliku, to całą ciężką pracę można wykonać, używając *pojedynczego wiersza* kodu Pythona:

```
swimmer, age, distance, stroke = fn.removesuffix(".txt").split("-")
```

To całkiem potężny wiersz kodu. Zastanawiamy się, co myśli trener...



Zaraz zaczyna się sesja treningowa, więc muszę lecieć. Wygląda świetnie! Mam dane z nazwy pliku, teraz potrzebuję przetworzenia danych z niego. Czas sięgnąć głęboko i zacząć działać, co nie?

**Gdy zostaniesz trenerem,
pozostaniesz nim na całe życie!**

Tak, zadanie nr 1 mamy z głowy.

Poświęćmy chwilę, aby przypomnieć sobie, z czym wiąże się zadanie nr 2 (na następnej stronie).

Zadanie nr 2: przetworzenie danych z pliku

Na pierwszy rzut oka wygląda na to, że jest tu trochę pracy. Ale nie martw się: w następnym rozdziale zajmiemy się wszystkimi tymi podzadaniami:

- 2a Odczytać wiersze tekstu z pliku.
- 2b Zignorować drugi wiersz tekstu.
- 2c Podzielić pierwszy wiersz tekstu w miejscach wystąpienia znaku „;” w celu uzyskania listy czasów.
- 2d Przekształcić każdy z czasów z formatu tekstowego „minuty:sekundy,setnasek” na liczbę.
- 2e Obliczyć średnią wszystkich czasów i skonwertować ją na format „minuty:sekundy,setnasek” (w celu wyświetlenia).
- 2f Wyświetlić zmienne z zadania nr 1, a następnie listę czasów oraz obliczoną średnią (z zadania nr 2).



*Kawa, ciasto *i*
podsumowanie rozdziału!
Czy może być coś
lepszego?!?*

Masz wystarczająco dużo czasu, aby przed rozpoczęciem lektury rozdziału trzeciego zrobić kilka rzeczy. A w ramach *wolnego czasu* zaparz sobie filiżankę ulubionego naparu, zjedz kawałek ciasta, przeczytaj podsumowanie rozdziału...

CELNE SPOSTRZEŻENIA

- W Pythonie łańcuchy znaków nie są zwykłymi łańcuchami znaków, ale **obiektami**, które mają mnóstwo wbudowanych możliwości.
- Atrybuty obiektu (w tym listę metod, które można wywołać na rzecz danego obiektu) można wyświetlić za pomocą podwójnego mamba `print dir`.
- Jeśli dysponuje się nazwą obiektu, użycie znanej notacji z kropką umożliwia odwoływanie się do dowolnego atrybutu tego obiektu, na przykład: `fn.upper`. Jeśli masz do czynienia z metodą, możesz ją wywołać, dodając parę nawiasów, na przykład: `fn.upper()`.
- Łańcuchy znaków udostępniają wiele metod, ale naszą ulubioną jest `split`, która, choć operuje na łańcuchu znaków, po wywołaniu zwraca listę.
- Wywołania metod obiektów można łączyć w **sekwencję**, ale należy przy tym zachować ostrożność, aby uniknąć występowania błędu `AttributeError` (patrz poprzedni punkt).
- Gdy pojawiają się komunikaty o błędach Pythona, zawsze czytaj je **od dołu do góry**.
- Sekwencja wywołań metod `removesuffix` i `split` to potężna kombinacja.
- Wielokrotne przypisywanie (określane także jako **rozpakowywanie**) to potężna technika, która pozwala, w ramach jednej instrukcji przypisania, określać wartości dowolnej liczby zmiennych. Można uznać, że to jedna z **supermocy** Pythona... i faktycznie tak jest.

Skorowidz

A

adres URL, 259, 276
 parametry, 262
 aktualizacja
 aplikacji internetowej, 522, 565
 danych, 398
 aplikacja internetowa, 247
 aktualizacja, 522, 565
 testowanie, 525, 568
 wyświetlenie wykresu, 534
 arkusz stylów CSS, 305, 306, 331
 atak typu SQL Injection, 454
 atrybuty, 38, 50, 51
 obiektu, 76

B

baza danych, 431
 aktualizacja, 481, 482
 dane uwierzytelniające, 563
 dodawanie danych, 453
 integracja, 510, 518, 545
 określenie struktury, 437
 pobieranie danych, 487
 tworzenie tabel, 446
 w serwisie PythonAnywhere, 562
 BeautifulSoup, 340
 białe znaki, whitespaces, 58
 biblioteka
 BeautifulSoup, 340
 datetime, 559
 enum, 120
 gazpacho, 342
 json, 386
 numpy, 429
 pandas, 409
 pathlib, 83
 random, 10
 standardowa PSL, 4, 30–34, 38
 moduły, 32
 szablonów Jinja2, 302
 requests, 340
 unittest, 580
 zipfile, 79
 błąd
 404, Not Found, 288, 294
 500, Internal Server Error, 263, 569

AttributeError, 65, 76
 KeyError, 271
 NameError, 263, 266, 267
 ValueError, 153

D

dane tabelaryczne, 407
 DBcm, 440
 import modułu, 441
 instalacja modułu, 440
 definiowanie
 funkcji, 38
 zmiennej, 10
 dekorator, decorator, 255, 583
 @app, 257, 300, 302
 dokumentacja
 metody, 57, 67
 Pythona, 36
 dołączanie
 arkuszy stylów, 306, 309, 331
 bibliotek, 5
 dostęp do elementu listy, 85
 dziedziczenie, 280, 281, 284, 302
 dzielenie łańcuchów znaków, 58
 dziennik
 błędów, 570
 dostępu, 570
 serwera, 570

E

edytor tekstu, 8
 edytory dla programistów, 587

F

Flask, 248
 instalacja, 248
 katalog aplikacji, 250
 komunikacja z przeglądarką, 259
 MDA, 251
 obsługa sesji, 269, 270
 parametryzowanie adresów
 URL, 262
 pobieranie danych z formularzy, 289
 silnik szablonów, 278
 testowanie aplikacji, 254, 271

tryb debugowania, 264, 302
 tworzenie aplikacji, 255
 uruchamianie serwera, 255
 f-łańcuchy, 181–186, 214, 419
 format
 JSON, 384–386
 datetime, 385
 SVG, 173
 formatowanie
 f-łańcuchów, 212
 łańcuchów znaków, 211, 214
 formularze, 289–302
 framework Flask, 248
 funkcja, 53
 add, 22
 convert2range, 191, 192, 214
 dir, 20, 38
 display_swimmers, 529–532
 draw, 6, 17, 25–27
 dump, 385
 enumerate, 165, 420
 get, 343
 hash, 55
 head, 430
 help, 21, 24, 57, 67
 index, 282
 int, 95, 117, 120
 len, 19, 27, 38, 223, 343
 list, 366, 543
 listdir, 452
 max, 191
 name, 255
 open, 79, 406
 print, 6, 20, 38, 105
 print(dir()), 50
 range, 38
 read_html, 411, 430
 remove, 24
 render_template, 275, 302
 replace, 419
 round, 111, 117, 120, 211
 set, 19, 38
 set_index, 416, 417, 430
 sorted, 221, 234
 str, 112, 117, 120, 257, 260
 to_dict, 412, 413, 430
 type, 19, 25, 38
 zip, 366, 543

funkcje, 136, 137
 definiowanie, 206
 parametry domyślne, 295
 przekazywanie argumentów, 286
 tworzenie, 124
 wbudowane, BIF-y, 4, 19

G

gazpacho, 370
 instalowanie pakietu, 342
 konfigurowanie, 370
 pobranie kodu HTML, 343
 porównanie z pandas, 424
 ustawienia domyślne, 355
 wyodrębnianie tabel HTML, 354
 generowanie
 kodu SVG, 187, 538
 liczb losowych, 5
 grafika wektorowa skalowalna, 174

H

hierarchiczna struktura danych, 406
 HTML, 173
 HTTPS, 327

I

instalowanie
 Flaska, 248
 gazpacho, 342
 modułu DBcm, 440, 566
 pandas, 409
 Pythona, xxxii
 VS Code, xxxiii
 instrukcja
 endblock, 281
 extends, 302
 if, 28, 38, 155–157, 160, 582
 if/else, 163, 167
 import, 128, 131, 132, 167, 503
 match, 582
 return, 129, 130, 167
 switch, 582
 with, 80, 81, 120, 441
 instrukcje Jinja2, 280, 302
 integracja bazy danych, 510, 518, 545
 interpolacja, 214
 interpreter, 14, 23

J

język SQL, 434
 Jinja2, 278, 279, 302
 instrukcje, 280, 302

szablony, 280, 282, 283
 wyrażenia, 280, 302
 JSON, 384–386
 Jupyter Notebook, 8, 429

K

katalog __pycache__, 142
 klasa, 576
 dataclass, 576
 UseDatabase, 442, 484
 klucz-wartość, 227, 244
 komentarz jednowierszowy, 96
 kompilowanie kodu, 14
 komunikat
 active, 265
 o błędzie, 64, 76, 152, 263
 PIN, 265
 stat, 265
 konkatencja łańcuchów znaków, 113, 181
 konwersja
 łańcucha, 94
 średniej, 111
 konwersje automatyczne, 380
 konwertowanie
 listy na zbiór, 219
 na liczby, 96
 wartości liczbowych, 192
 krotka, tuple, 18, 38, 137, 148, 167, 246, 484

L

lista, list, 18, 38, 58, 85, 120, 148, 167, 226, 246
 imion pływaków, 217
 liczb, 77
 lista-zbiór-lista, 219, 244
 plików, 121, 452
 pusta, 104, 120, 232
 rozwijana, 277
 dynamiczne tworzenie, 284
 składana, list comprehension, 485, 491–494, 514, 546
 służąca do filtrowania, 546
 służąca do przekształcania, 546
 testowanie, 516
 listy
 metody, 105
 sortowanie, 145
 tworzenie, 104
 usuwanie powtórzeń, 219
 literały łańcuchowe, 169

Ł

łańcuch znaków, 50, 51, 55
 dzielenie, 58
 jako obiekt, 51, 76
 konkatencja, 181
 konwersja na liczbę, 94, 96
 metody, 66, 76
 słowo kluczowe in, 98, 158
 użycie metody split, 58
 wyszukiwanie znaku, 158

M

manipulowanie danymi, 371
 MariaDB, 547, 549, 550
 konfiguracja, 551
 migracja danych, 552, 556
 testowanie aplikacji, 560
 zgodność zapytań, 558
 zmiana kodu narzędzi, 559
 menedżer kontekstu, context manager, 583
 metoda, 50, 53
 append, 105, 120, 235
 execute, 444, 484
 fetchall, 444, 445, 468, 484
 fetchmany, 484
 fetchone, 445, 484
 find, 352, 355, 356, 370
 fn.lower, 54
 fn.upper, 54
 HTTP GET, 254–258, 302
 HTTP POST, 289, 302, 528
 len, 149
 readlines, 82
 remove, 149
 removesuffix, 67, 68, 92, 378
 replace, 382
 reverse, 204, 214
 sort, 145, 167
 split, 58–60, 76, 93, 313
 strip, 93
 metody, 50, 53
 listy, 105
 łączenie wywołań, 65
 obiektu łańcuchów znaków, 66
 wywoływanie, 59
 miejsce wprowadzania, focus, 10
 migracja danych, 552, 556
 moduł
 datetime, 98
 DBcm, 440, 484
 enum, 83
 gazpacho, 343
 json, 385
 os, 167
 pprint, 240, 244

pyrg_funkcje, 192, 214
 statistics, 110, 120
 time, 98
 webbrowser, 199, 203, 214
 modyfikowanie kodu, 55, 309

N

narzędzia
 do formatowania kodu, 587
 konwersji, 540
 narzędzie
 danych, 506–508
 do aktualizacji bazy danych, 481, 482
 pip, 248
 REPL, 14, 50
 sqlite3, 555
 nawiasy
 klamrowe, 184, 229, 230, 244
 podwójne, 280, 284, 316
 kwadratowe, 58, 85, 120, 138, 229, 244
 rozszerzenie notacji, 223, 347, 370
 nazwa pliku, 56
 notacja
 wycinków, 346
 z kropką, 128
 notatnik, 8

O

obiekt, 50, 51
 łańcucha znaków
 metody, 66
 pliku, 81, 120
 request.form, 290, 302
 obsługa
 HTTPS, 327
 sesji, 269, 270
 odczytywanie z pliku, 116, 200
 odwzorowania, mappings, 546
 operator
 dzielenia całkowitego, //, 112, 120
 dzielenia, /, 112
 in, 167
 kropki, 53, 59
 morsa, walrus operator, 581
 przypisania, =, 50, 71, 72
 operatory standardowe, 120

P

pakiet
 attrs, 576
 bs4, 340
 gazpacho, 340
 lxml, 340
 requests, 340

pandas, 409, 429, 430
 import, 410
 instalacja, 409
 ramki danych, 411
 para klucz-wartość, 227, 244
 parametry domyślne, 295, 300, 302
 parsowanie kodu HTML, 340, 354, 370
 pętla
 for zagnieżdżona, 373
 for, 15, 38, 98–101, 220, 464, 478
 while, 100, 101
 plik
 .DS_Store, 146, 148
 app.py, 306
 CSS, 307
 index.html, 306
 schema.sql, 553
 WSGI, 326
 pliki
 .css, 306
 .htm, 201
 .ipynb, 26
 .py, 26
 .txt, 202, 314
 operacje, 80
 otwieranie, 81, 116
 przetwarzanie danych, 76, 78
 tryb do odczytu, 389
 tryb do zapisu, 406
 wyodrębnianie danych z nazwy, 50
 zamykanie, 81, 116
 pobieranie strony HTML, 343
 podpowiedzi typów, 585
 podwójne podkreślenia, 52
 polecenie
 mysqldump, 564
 pragma table_info, 474
 source, 564
 source bd.sql, 567
 SQL
 create table, 446, 553, 557
 delete, 457, 459, 469, 484
 describe, 555
 insert, 453, 458, 460, 471, 473, 484
 select, 469, 473, 484
 show tables, 554
 source schema.sql, 554
 update, 484
 SQLite pragma table_list, 442
 unzip, 325
 program sglitertools win32-x86, 552
 przeglądarka Firefox, 386
 przekazywanie argumentów do
 funkcji, 286
 przypisywanie, 71–73
 PSL, Python Standard Library, 30

PyPI, Python Package Index, 4, 34, 38
 Python, 2
 instalacja, xxxii
 PythonAnywhere, 318, 320, 323, 331
 instalacja DBcm, 566
 kopiowanie do bazy danych, 567
 planowanie aktualizacji, 400
 testowanie skryptu, 404
 tworzenie bazy danych, 562

R

ramka danych, dataframe, 411, 430
 konwersja na słownik, 413, 416, 417
 usuwanie danych, 414
 wybór kolumn, 412
 refaktoryzacja, 507
 referencje do obiektów, 106
 relacja dziedziczenia, 281, 284
 REPL, 14, 50
 request.form, 290, 302
 rozpakowywanie, 70–73, 76

S

scrapowanie, 338
 sekwencja, 138
 separator, 58, 314
 sformatowane literały łańcuchowe,
 Patrz f-łańcuchy
 skalowanie liczb, 192
 skróty klawiszowe, 39
 Shift+Enter, 10
 słownik, dictionary, 215, 226, 227,
 244, 246
 obsługa konwersji, 378
 przetwarzanie, 241
 słowników, 374, 409, 418
 tworzenie, 230
 wyświetlanie, 239
 zastosowanie, 270, 290, 337, 376
 zwracanie listy kluczy, 375, 419
 słowo kluczowe
 as, 82, 84
 class, 577
 def, 38, 124, 167, 206
 import, 5
 in, 27, 28, 38, 159
 self, 577
 with, 80, 81, 120
 sortowanie
 listy, 145
 słownika, 234
 specyfikator formatu, 211, 214
 SQL, 434
 SQLite, 434, 484, 550

struktura danych, 4
 krotka, 18
 lista, 18, 145
 słownik, 18
 tablica, 17
 zbiór, 18

SVG, Scalable Vector Graphics, 173
 sygnatura funkcji, 124
 symbole
 zastępcze, 453, 455, 484, 558, 572
 szablon
 base.html, 281
 bazowy, 280, 331
 Jinja2, 279–283, 526
 generowanie kodu SVG, 538

Ś

ścieżka, 259
 ślad wykonania, traceback, 64
 średnia arytmetyczna, 110
 środowisko wirtualne, virtual environment, 586

T

tabela wyszukiwania, 337
 tabele bazy danych, 437
 tablica, 17, 18
 przeglądowa, lookup table, 231, 371
 testowanie, 580
 aplikacji internetowej, 331, 525
 zapytań, 496, 503
 list składanych, 516

tworzenie
 aplikacji internetowej, 247, 297
 bazy danych, 562
 dynamiczne listy rozwijanej, 284
 f-łańcucha, 195, 197, 214
 funkcji, 124, 206, 504
 listy, 104
 listy nazw plików, 217
 modułu, 134, 167
 obiektu pliku, 81
 obiektu łańcucha znaków, 50
 słownika, 230
 szablonów Jinja2, 279
 tabel bazy danych, 437, 446
 wykresu słupkowego, 174
 zmiennej, 50, 70

U

uruchamianie kodu, 10
 usługa PythonAnywhere, 331

V

VS Code
 instalacja, xxxiii
 uruchomienie, 7
 utworzenie notatnika, 8

W

wartości domyślne, 296, 302
 wartość
 __main__, 256
 logiczna
 False, 161
 True, 161
 wątki, threads, 584
 wdrażanie aplikacji
 w chmurze, 303–327, 331
 w serwisie PythonAnywhere, 323, 547, 561
 wdrożenie łańcucha wywołań, 316
 webscraping, 333
 widoczność zmiennej, 268
 wielokrotne przypisywanie, 70–73, 76
 WSGI, Web Server Gateway Interface, 326, 331
 współbieżność, 584
 wstrzykiwanie kodu SQL, 454
 wycinek, 223, 345–353, 370
 składnia, 350
 wyjątki, 579
 wykres słupkowy, 170, 544
 dodawanie rekordów, 392
 tworzenie, 174, 296
 zapisywanie, 296
 wyrażenia Jinja2, 280, 302
 wyrażenia przypisania, assignment expression, 581
 wyszukiwanie, 28
 tabel HTML, 354
 w słowniku, 237, 238
 wartości, 158
 wierszy tabeli, 356
 znaczników, 355
 znaku, 158
 wyświetlanie
 metod listy, 105
 słownika, 239
 wykresu słupkowego, 203, 208, 534
 wywołanie .strip().split(), 120
 wywoływanie
 łączone metod, 65
 metody, 59

Z

zakresy widoczności, 268
 zaokrąglenie wartości, 210, 211
 zapisywanie
 do pliku, 116, 200
 kodu HTML, 199
 zbiór, set, 19, 38, 246
 zestaw, suit, 88
 zintegrowane środowisko
 programistyczne, IDE, 587
 zliczanie liczby znaczników, 360
 złożone struktury danych, 244
 zmienna, 6, 50
 __name__, 256
 tablicowa, 6
 zmienne
 dynamiczne, 84
 globalne, 268, 302
 lokalne, 268
 modyfikowalne, 55
 niezmiennie, 55
 tworzenie, 50, 70
 znacznik
 <body>, 199

, 176
 <h3>, 178
 <html>, 199
 <link>, 307, 310
 <option>, 316
 <p>, 307
 <rect>, 188, 190
 <select>, 274, 277, 285
 <svg>, 187, 188, 190, 198
 <table>, 360, 411
 <td>, 360
 <th>, 362
 <title>, 178
 <tr>, 359, 364
 znaczniki czasu, 479
 znak
 @, 255, 259
 apostrof, 51
 cudzysłów, 50
 dwukropek, 26, 88, 120, 158, 211, 233
 kropka, 552
 negacji, ~, 415
 nowego wiersza, \n, 91, 92
 podkreślenia, 52
 przecinek, 58
 wykrzyknik, 497

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 

Python jest wyjątkowy! Umożliwia nie tylko tworzenie rozbudowanych aplikacji, ale również rozwiązywanie złożonych problemów. Korzystają z niego programiści, analitycy danych, naukowcy, inżynierowie, specjaliści od sztucznej inteligencji i profesjonalści z wielu innych dziedzin. Przystępność i uniwersalność Pythona sprawiają, że jest jednym z najchętniej używanych języków programowania. Jeśli przed zagłębieniem się w tajniki kodowania powstrzymywała Cię obawa przed nudnym wertowaniem nieciekawych podręczników, to właśnie trzymasz w rękach książkę, która jest dla Ciebie!

Zacznij od arkusza kalkulacyjnego udającego „aplikację” przeznaczoną dla jednego użytkownika...



...i przekształć go w napisaną w Pythonie aplikację internetową, z której może korzystać dowolna liczba osób.

Ta pozycja, podobnie jak inne z serii **Rusz głową!**, została przygotowana zgodnie z jedyną w swoim rodzaju metodyką nauczania, wykorzystującą zasady funkcjonowania ludzkiego mózgu. Dzięki zagadkom, tajemniczym historiom, angażującym ćwiczeniom i przystępnie podanej wiedzy bez trudu przyswoisz nawet dość złożone koncepcje, takie jak programowanie zorientowane obiektowo, aplikacje sieciowe czy uczenie maszynowe. Znajdziesz tu zabawne i niekonwencjonalne ilustracje, świetne analogie, a w toku nauki krok po kroku zbudujesz własną aplikację. Przekonasz się, że to absolutnie wyjątkowy i niezwykle skuteczny podręcznik!

Jeśli Python jest na Twojej liście rzeczy do zrobienia, to zacznij od tej książki!

Daniel Hinojosa
programista, instruktor,
prezenter

Paul Barry – jest autorem książek technologicznych i trenerem. Specjalizuje się w języku Python. Słynie z jasnego i zwięzłego stylu pisania, a także z umiejętności przekazywania złożonych idei w przystępny sposób. Na co dzień wykłada na wydziale informatyki South East Technological University. Mieszka w Irlandii, w mieście Carlow.



KOD KORZYŚCI
Sięgnij po więcej! ▶





helion.pl



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

ISBN 978-83-289-0700-3



9 788328 907003

Cena: 129,00 zł