O'REILLY®

Wydanie III

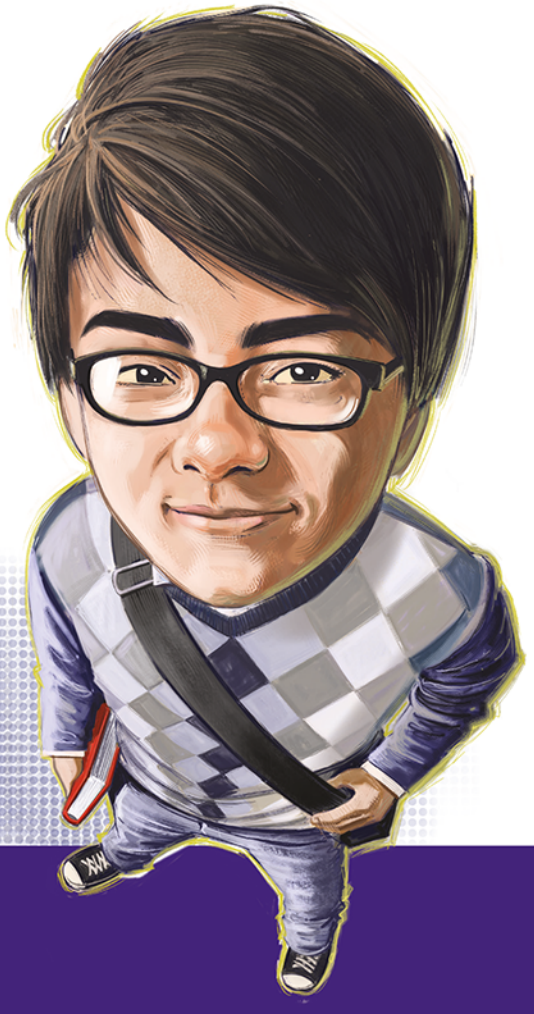
Head First

Python

Rusz głową!

Przewodnik
dla uczących się
podstaw Pythona

Paul Barry



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

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.

Spis treści (podsumowanie)

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?

O 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 zbytniego 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.

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**. Zaczniesz od łańcuchów znaków w Pythonie, ucząc się, jak nimi manipulować, jednocześnie cały czas będziesz pracować nad rozwiązaniem problemu trenera. Dodatkowo przyjrzysz 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!)...

2 Listy liczb
Przetwarzanie danych z list

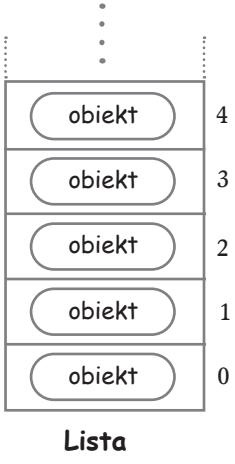
Im więcej kodu piszesz, tym lepszy się stajesz. To takie proste.

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. Napiłeś się już kawy i zjadłeś ciasto, więc czas zakasać rękawy i wrócić do pracy.

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 wspominana 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 udostępniać**. 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

4

Sformatowane literały łańcuchowe

Twórz wykresy na podstawie danych

Czasami to najprostsze metody pozwalają wykonać zadanie.

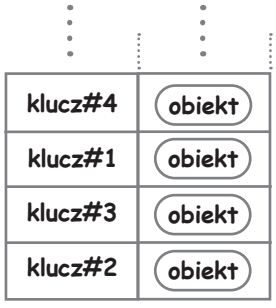
W tym rozdziale zajmiemy się w końcu tworzeniem wykresów słupkowych dla trenera. Zrobimy to, używając wyłącznie łańcuchów znaków. Już wiesz, że łańcuchy znaków w Pythonie są pełne **wbudowanych dobroci**, a **sformatowane literały łańcuchowe** Pythona, znane również jako *f-łańcuchy*, zwiększają jeszcze 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!

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!



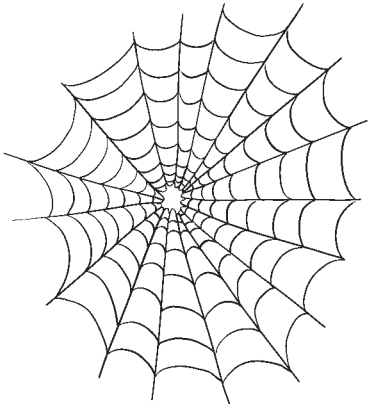
Słownik

Tworzenie aplikacji internetowej

6 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!



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!

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*...

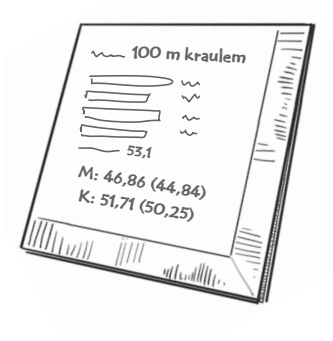
9

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 konwersje, integracje, aktualizacje, wdrożenia i wiele więcej. A na końcu tego wszystkiego serwetkowa specyfikacja trenera stanie się *rzeczywistością*. Nie możesz się doczekać? My też... zatem do dzieła!



9½

Praca ze ~~słowami~~ ramkami danych

Dane tabelaryczne

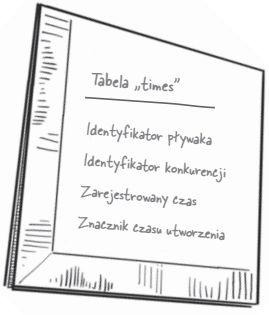
Czasami wygląda to tak, jakby wszystkie dane świata chciały znaleźć się w tabeli.

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.

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, **wybierzesz** i **usuniesz** je z niej, wykonując wszystkie te zadania z poziomu kodu Pythona za pomocą odpowiednich zapytań SQL.



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 zaczynajmy. Przed nami wiele pracy związanej z **integracją**.


```
[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)
```

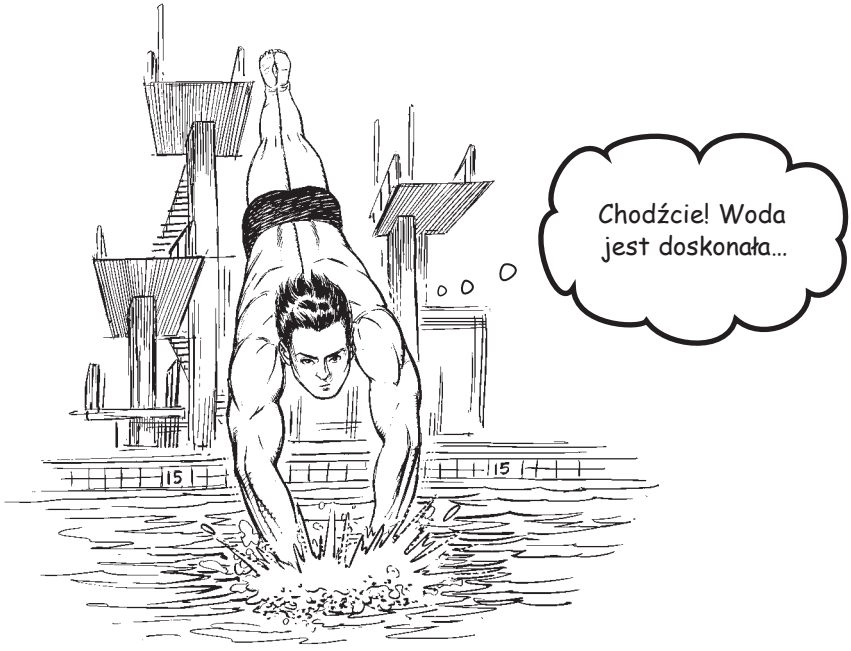
Dodatek. Pozostałości

Dziesięć najważniejszych zagadnień, o których nie napisaliśmy

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

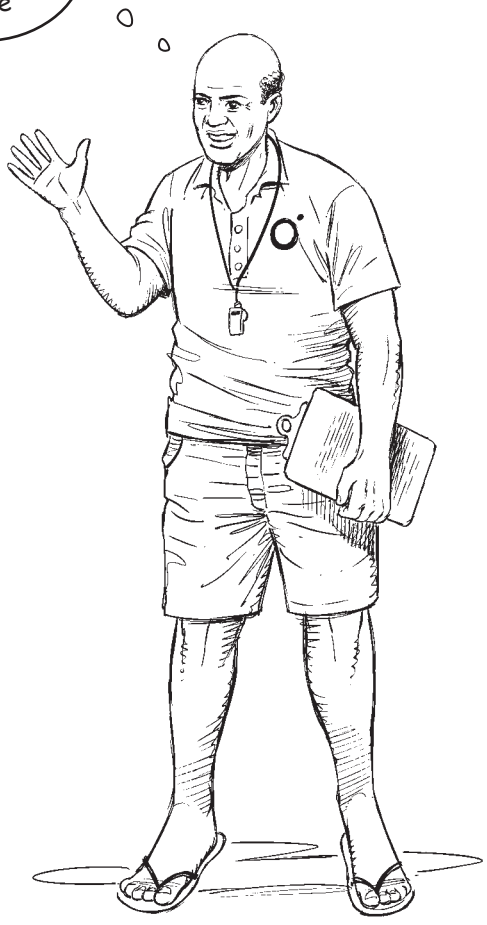
Zwłaszcza gdy autor pochodzi z *Irlandii*, narodu słynącego z osób, które nie do końca wiedzą, kiedy przestać mówić 😉 A my uwielbiamy rozmawiać o Pythonie, naszym ulubionym języku programowania. W tym „Dodatku" przedstawiamy dziesięć najważniejszych zagadnień, o których z radością byśmy Ci opowiedzieli, gdybyśmy mieli do dyspozycji kolejnych czterysta stron. Są tu informacje o klasach, obsłudze wyjątków, testowaniu, morsie (serio? o *morsie*? tak: o morsie), instrukcjach `switch`, 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!

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**. Zaczniesz od łańcuchów znaków w Pythonie, ucząc się, jak nimi manipulować, jednocześnie cały czas będziesz pracować nad rozwiązaniem problemu trenera. Dodatkowo przyjrzysz 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...

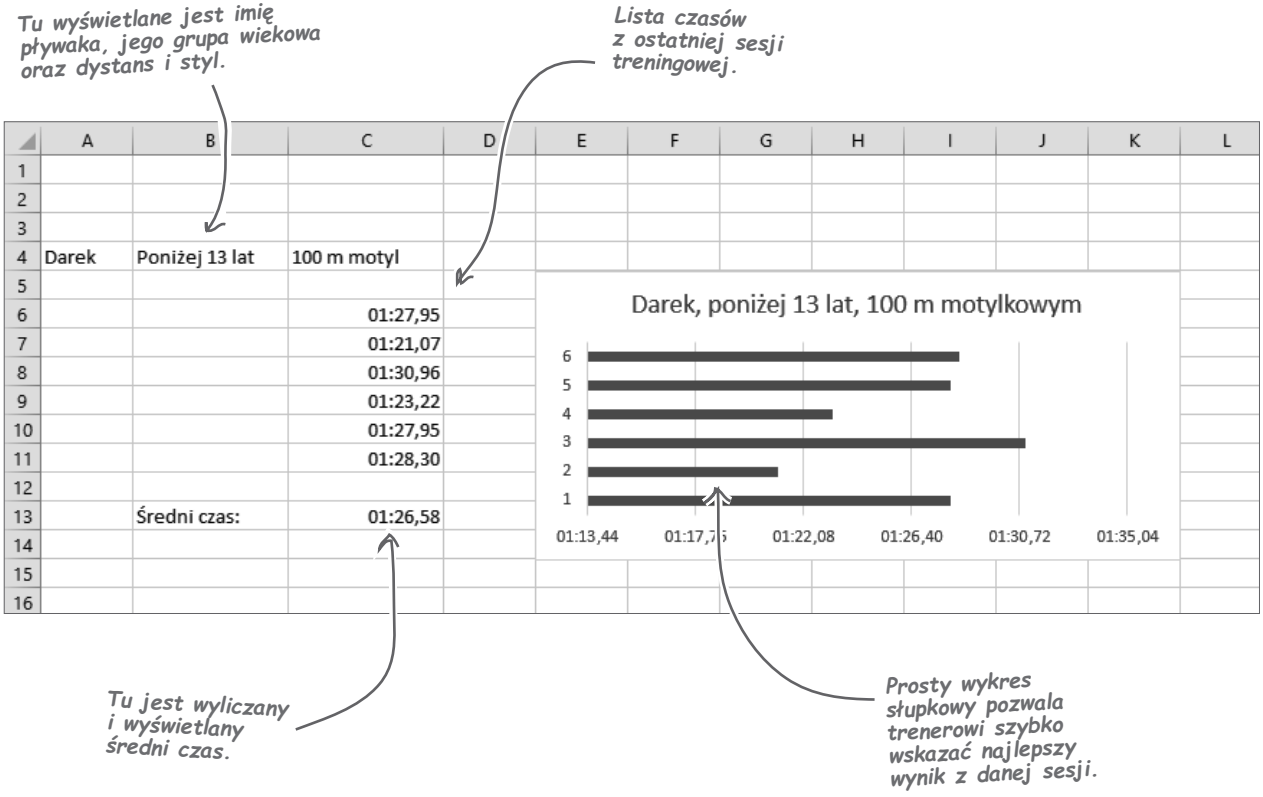
Jeśli chodzi o monitorowanie postępów młodych pływaków, trener wszystko robi *ręcznie*. Podczas sesji treningowej zapisuje czasy uzyskiwane przez poszczególnych pływaków na kartkach, a następnie w domu *ręcznie* wpisuje czasy każdego pływaka do swojego ulubionego arkusza kalkulacyjnego, aby przeprowadzić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



	A	B	C
4	Darek	Poniżej 13 lat	100 m motyl
6			01:27,95
7			01:21,07
8			01:30,96
9			01:23,22
10			01:27,95
11			01:28,30
13		Średni czas:	01:26,58

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

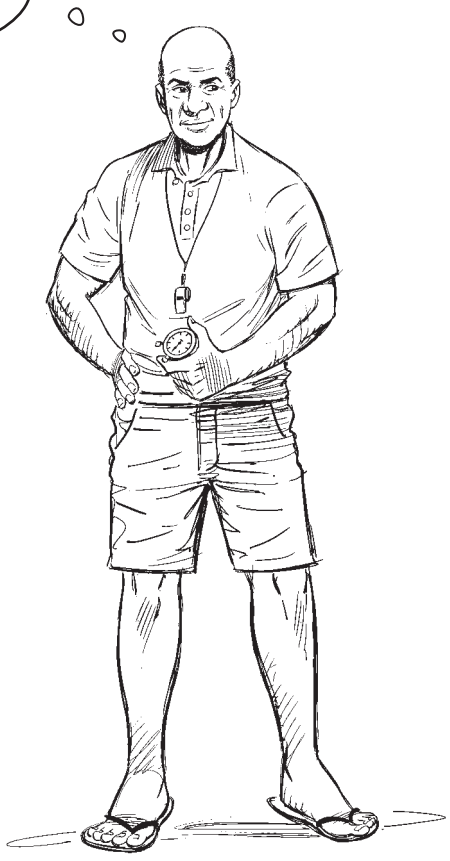
```
1  1:27,95;1:21,07;1:30,96;1:23,22;1:27,95;1:28,30
2
```

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

Poświęć chwilę na uważne przejrzenie 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?



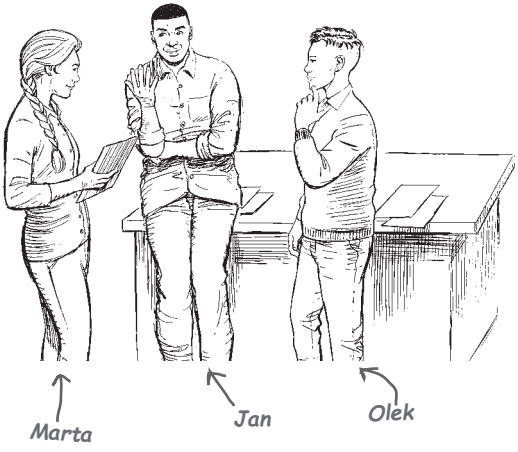
Trener uwielbia swój nowy stoper i planuje używać go podczas wszystkich kolejnych sesji treningowych.

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?*

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-motyl” 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.

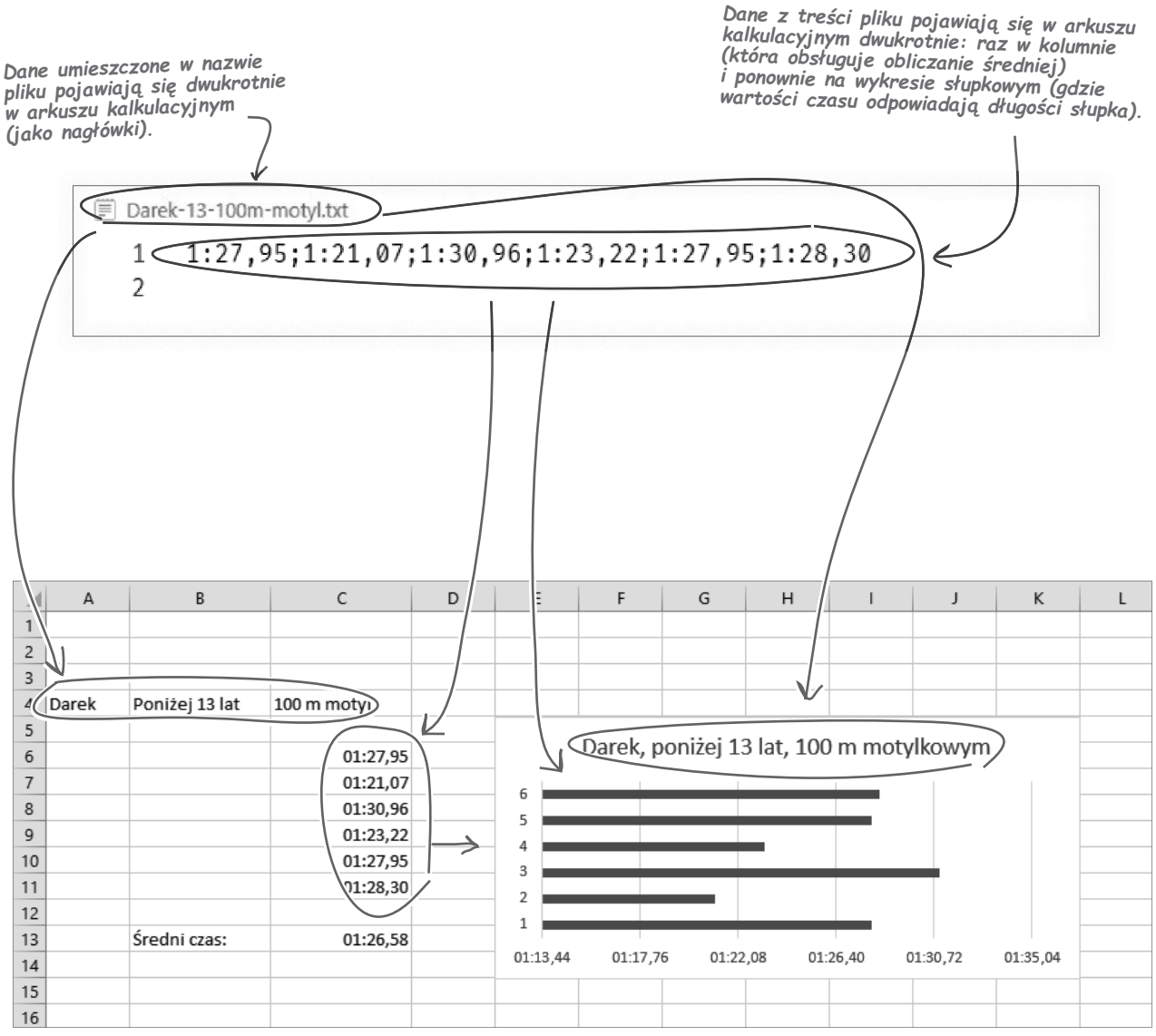
d) Przekształć każdy z czasów z formatu tekstowego „minuty:sekundy,setnesek" na liczbę.

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:



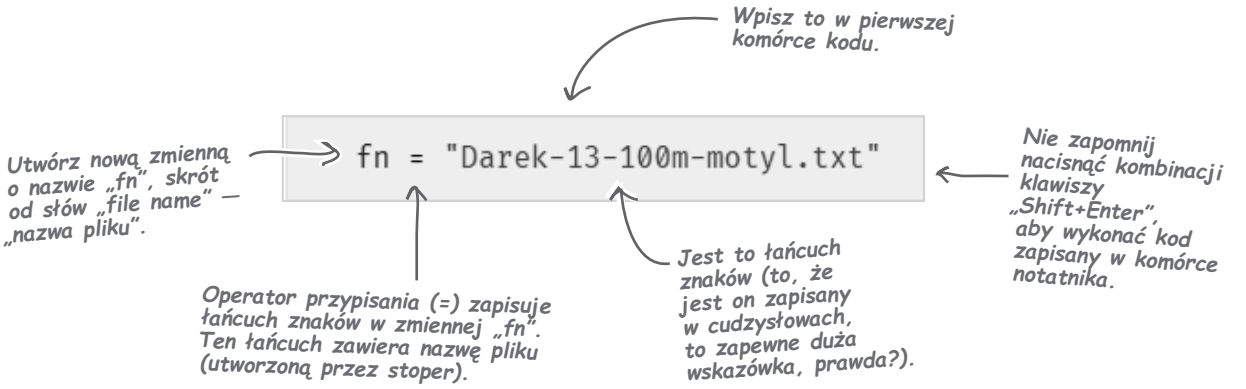
Nasze pierwsze zadanie: wyodrębnić dane z nazwy pliku

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*.



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` 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`, zobaczmy 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__', '__getattribute__', '__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?

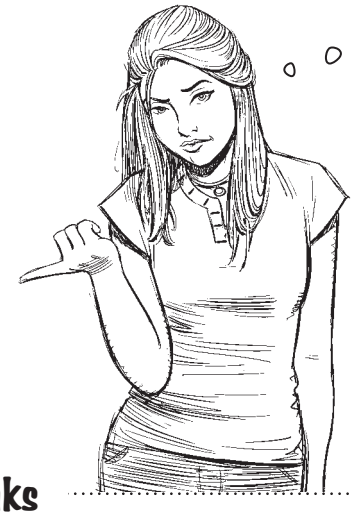
Daj sobie chwilkę, by docenić to, na co teraz patrzysz

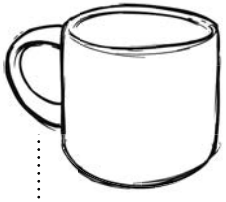
Jeśli patrzysz na tę długą listę atrybutów i myślisz, że jest tu strasznie dużo do nauczenia się, to zastanów się nad tym: w łańcuchy znaków jest wbudowanych bardzo dużo funkcjonalności, dla których *nie musisz* pisać kodu.

Nie istnieją głupie 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.





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 anglojęzycznego 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 helloł... pisanie o „dunderach" byłoby zbyt niepoważne nawet jak na TĘ KSIĄŻKĘ!

Zaostrz 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.

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.

Pamiętaj, by nacisnąć kombinację klawiszy „Shift+Enter", aby wykonać każdą komórkę.

```
fn.upper()
```

```
'DAREK-13-100M-MOTYL.TXT'
```

Metoda „upper" zwraca kopię łańcucha znaków zapisaną DUŻYMI LITERAMI.

```
fn.lower()
```

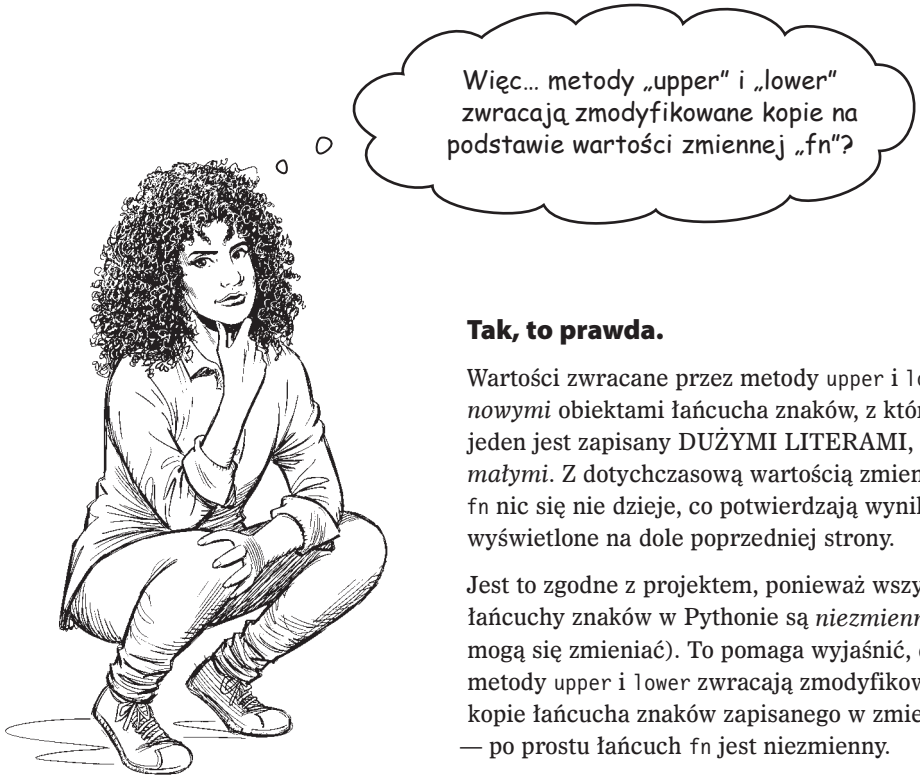
```
'darek-13-100m-motyl.txt'
```

Metoda „lower" zwraca kopię łańcucha znaków zapisaną małymi literami.

```
fn
```

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

Pierwotna wartość zmiennej „fn" nie uległa zmianie, ponieważ powyższe metody zwracają zmodyfikowaną kopię łańcucha znaków.



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 *niezmienne* (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 niezmienne, 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 przechwytującego 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:

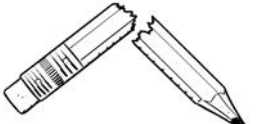
1a **Pobrać nazwę pliku.** ✔

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!



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

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).

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 szklankę 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 niezmienne, 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 „-".

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-motyl.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 ołówek

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.

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

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

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

Fragment przed „13".

Fragment po „13".

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-motyl.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 obok siebie.

```
['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 jako separatora znaku „.” (czyli kropki)…?

Nie. To działa inaczej!

O kurczę!

```
---------------------------------------------------------------------------
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 zacząłeś 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.

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. Zacznij 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

Przejrzyjmy 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',
'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'
```

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"`:

```
"motyl.txt".removesuffix(".txt")
```

```
'motyl'
```

Wszystkie cztery przypadki testowe działają zgodnie z oczekiwaniami. Wygląda na to, że „removesuffix" jest metodą, której szukamy.

```
"kraul.txt".removesuffix(".txt")
```

```
'kraul'
```

Czy zauważyłeś, że „removesuffix" zwraca łańcuch znaków (a nie listę)? Ma to ważną implikację, ponieważ można połączyć metodę zwracającą łańcuch znaków z inną metodą zwracającą łańcuch (i uniknąć tego paskudnego błędu AttributeError).

```
"grzbietowy.txt".removesuffix(".txt")
```

```
'grzbietowy'
```

```
"klasyczny.txt".removesuffix(".txt")
```

```
'klasyczny'
```

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.

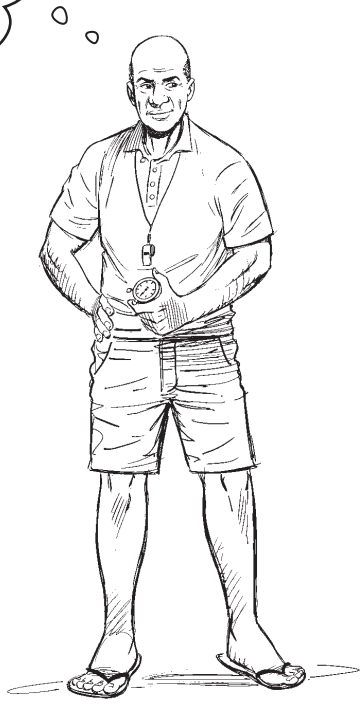
```
fn.removesuffix(".txt").split("-")
```

```
['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.

Nadszedł czas na postawienie parafki.

1a **Pobrać nazwę pliku.** ✔

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

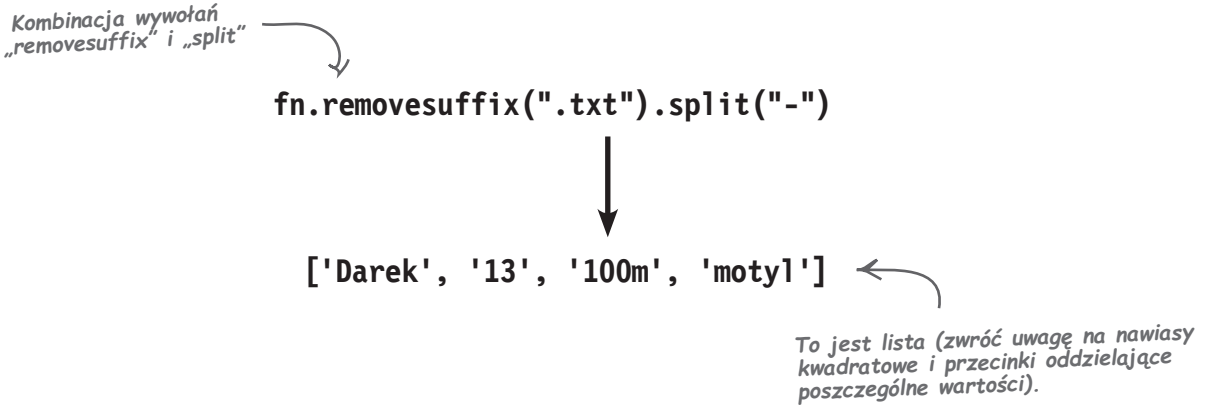
Nie wiemy jak Ty, ale my poczuliśmy się świetnie!

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?



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).

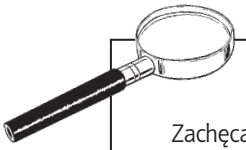
Wielokrotne przypisywanie (określane także jako „rozpakowywanie")

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.

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

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



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
```

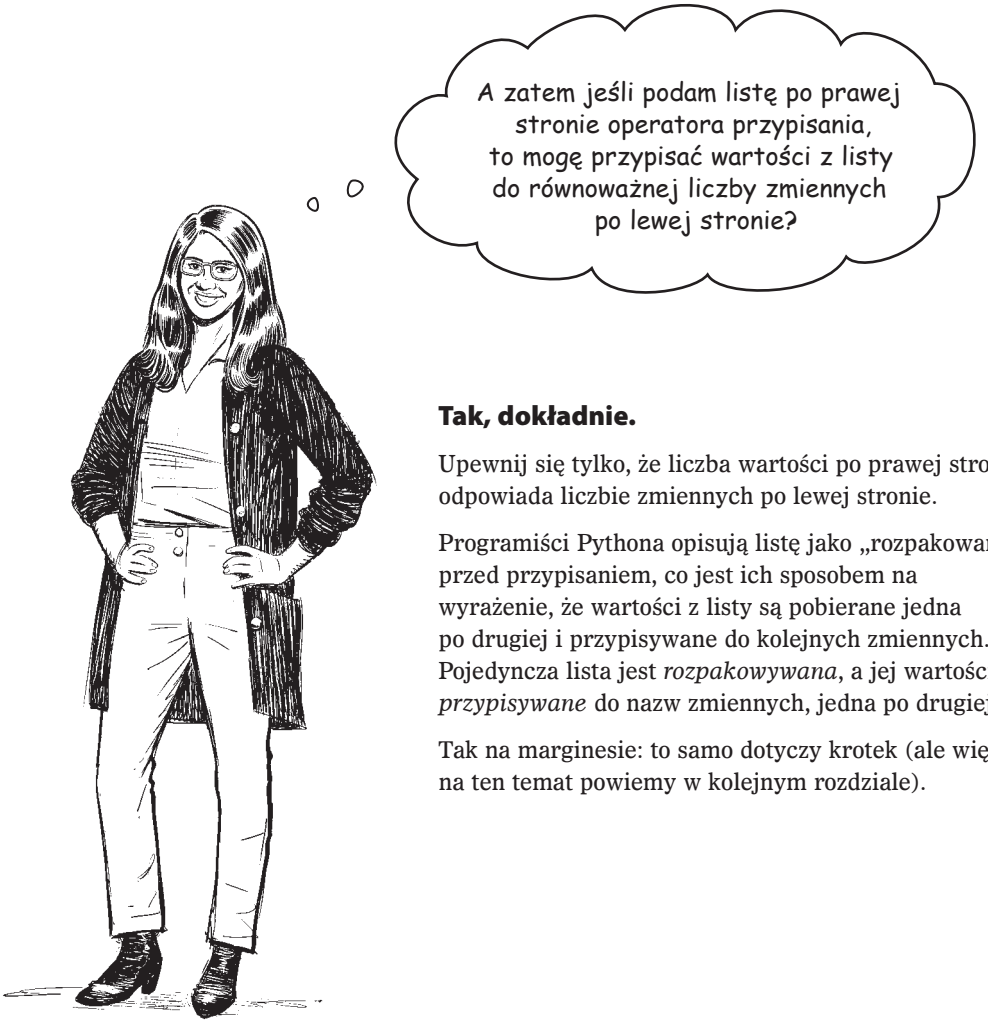
Kolejność zmiennych po lewej stronie operatora przypisania jest dopasowywana do wartości po prawej stronie.

```
meaning
```

```
42
```

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 umieszczania tej listy w nawiasach kwadratowych.



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 ołówek

Rozwiązanie ze strony 73.

Miałeś wziąć ołówek 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,setnesek" na liczbę.**
- **2e** **Obliczyć średnią wszystkich czasów i skonwertować ją na format „minuty:sekundy,setnesek" (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 mambo `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

G

H

I

J

K

L

Ł

M

Ś

T

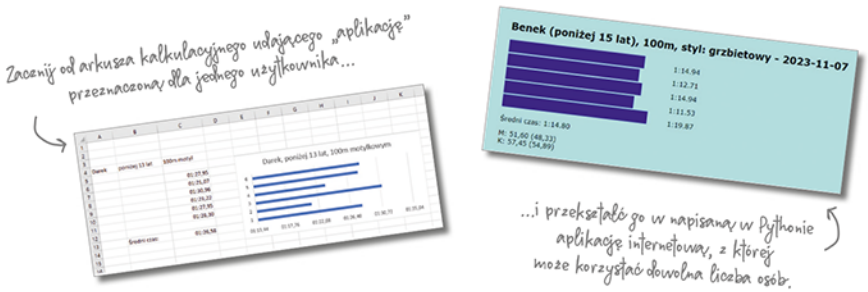
U

V

W

Z

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 profesjonaliści 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!



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 miasteczku Carlow.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-0700-3
9 788328 907003
Cena: 129,00 zł