O'REILLY®

Helion

Rusz głową!

Nauka programowania

Przewodnik
po kodowaniu
i myśleniu komputacyjnym

Uważaj na powszechnie występujące błędy i pułapki

Unikaj zawstydzających błędów składniowych

Rozpocznij karierę programisty

Gimnastykuj swój umysł, rozwiązując około 130 łamigłówek i ćwiczeń

Dowiedz się, dlaczego wszystko to, co Twoi znajomi wiedzą o informatyce, jest niezupełnie prawdziwe

Eric Freeman

Tytuł oryginału: Head First Learn to Code: A Learner's Guide to Coding and Computational Thinking

Tłumaczenie: Maksymilian Gutowski

ISBN: 978-83-283-4697-0

Helion SA
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/nauprg.zip

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

Printed in Poland.

Spis treści (streszczenie)

Spis treści (część właściwa)

Wstęp

Programowanie zaczyna się w głowie. Kiedy próbujesz się czegoś *nauczyć*, Twój *mózg* stara się wyświadczyć Ci przysługę, robiąc wszystko, co może, żebyś niczego *nie zapamiętał*. Twój mózg myśli sobie: „Lepiej zostawić w głowie miejsce na ważniejsze rzeczy, na przykład wiedzę o tym, przed jakimi dzikimi zwierzętami uciekać, albo o tym, że nie powinno się jeździć na desce snowboardowej w samych slipach". Jak zatem *przechytrzyć* mózg, żeby zaczął myśleć, że Twoje życie istotnie zależy od umiejętności programowania?

1 Myślenie komputacyjne

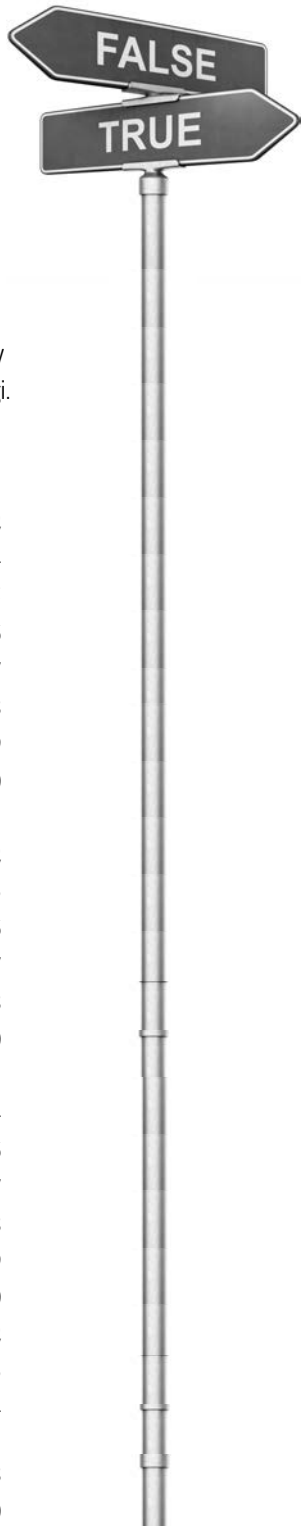
Zaczynamy

Umiejętność myślenia komputacyjnego daje Ci kontrolę. Jest tajemnicą poliszynela, że otaczający nas świat staje się coraz silniej połączony, konfigurowalny, programowalny i — co tu dużo mówić — **komputacyjny**. Możesz wobec tego albo pozostać biernym uczestnikiem tej rzeczywistości, albo *nauczyć się kodowania*. Umiejętność kodowania pozwoli Ci zostać twórcą czy też dowódcą — będziesz mógł powiedzieć wszystkim tym komputerom, co mają zrobić *dla Ciebie*. Kiedy umiesz kodować, masz kontrolę nad swoim losem (a już na pewno jesteś w stanie zaprogramować swój połączony z internetem system zraszaczy w ogródku). Jak się jednak nauczyć kodować? Przede wszystkim musisz nauczyć się **myśleć komputacyjnie**. Kolejnym krokiem jest zapoznanie się z **językiem programowania**, który pomoże Ci się dogadać ze swoim komputerem, telefonem lub dowolnym innym urządzeniem z procesorem w środku. Co zyskasz dzięki temu? Czas, władzę i więcej okazji do zajęcia się tymi wszystkimi twórczymi przedsięwzięciami, na których Ci zależy. Zacznijmy zatem...

Proste wartości, zmienne i typy

2 Znaj swoje wartości

Komputery radzą sobie dobrze tylko z dwiema rzeczami: przechowywaniem wartości i wykonywaniem na nich działań. Być może sądzisz, że robią wiele innych rzeczy, takich jak przesyłanie tekstu, wykonywanie transakcji handlowych, używanie Photoshopa lub pomaganie w kierowaniu samochodem. Tymczasem wszystkie czynności, jakie wykonują komputery, można rozłożyć na **proste działania** wykonywane na **prostych wartościach**. Elementem **myślenia komputacyjnego** jest nauczenie się wykorzystywania takich działań i wartości do tworzenia bardziej złożonych i zaawansowanych rzeczy — do czego dojdziemy już wkrótce. Najpierw jednak przyjrzymy się samym tym wartościom, działaniom, które można na nich wykonywać, oraz roli, jaką odgrywają w tym wszystkim **zmienne**.



Wartości logiczne, decyzje i pętle

3 Decyzyjny kod

Czy nie odnosisz wrażenia, że programy, które do tej pory napisaliśmy, nie są zbyt ciekawe? Innymi słowy, cały nasz kod składał się ze zbiorów instrukcji, które interpreter ewaluował z góry na dół — bez żadnych zwrotów akcji, niespodzianek czy niezależnego myślenia. Kod musi **podejmować decyzje**, **przejąć ster nad swoim przeznaczeniem** i wykonywać instrukcje **więcej niż raz**, aby był ciekawy. W tym rozdziale dowiesz się, jak o to zadbać. Przy okazji opowiemy sobie o tajemniczej chińskiej grze, poznamy pana Boole'a i zobaczymy, jak typ danych uwzględniający tylko dwie wartości jest godny uwagi. Dowiemy się też, jak poradzić sobie ze straszliwą **nieskończoną pętlą**. Zaczynamy!

4 Listy i iteracje

Odrobina struktury

Typy danych to nie tylko liczby, łańcuchy i wartości logiczne. Do tej pory pisałeś pythonowy kod z wykorzystaniem samych prymitywów — liczb zmiennoprzecinkowych, całkowitych, łańcuchów i wartości boolowskich — o wartościach w rodzaju `3.14`, `42`, `"pan tu nie stał"` czy `True`. Prymitywy pozwalają osiągnąć bardzo dużo, ale w końcu dojdziesz do etapu, na którym będziesz chciał stworzyć kod wykorzystujący liczne zbiory danych, obejmujące na przykład wszystkie elementy w koszyku zakupów, nazwy najjaśniej świecących gwiazd bądź całe katalogi produktów. Będziesz potrzebował do tego czegoś *mocniejszego*. W tym rozdziale przyjrzymy się **liście** — nowemu typowi, który umożliwia przechowywanie zbiorów wartości. Listy pozwalają na **ustrukturyzowanie** danych, dzięki czemu możesz uniknąć zaśmiecenia kodu milionami zmiennych o własnych wartościach. Dowiesz się też, jak traktować te wszystkie wartości jako całość, a także jak **iterować** listy, korzystając z pętli `for`, o której dowiedziałeś się w poprzednim rozdziale. Po przeczytaniu tego rozdziału będziesz umiał sprawniej posługiwać się danymi.

Funkcje i abstrakcje

5 Funkcyjny kod

Wiesz już bardzo dużo. Zmienne, typy danych, wyrażenia warunkowe, iteracja — to wszystko w zupełności wystarczy, aby napisać **praktycznie dowolny program**. Co więcej, informatyk mógłby Ci powiedzieć, że wystarczy to do napisania każdego programu, jaki ktokolwiek mógłby wymyślić. Nie spoczywaj jednak na laurach, ponieważ kolejnym krokiem w nauce myślenia komputacyjnego jest **tworzenie abstrakcji** kodu. Brzmi to pewnie skomplikowanie, ale w rzeczywistości chodzi właśnie o uproszczenie pracy z kodem. Abstrakcje są dźwignią ułatwiającą pisanie programów o coraz większej złożoności i mocy. Pozwalają na umieszczanie kodu w małych, schludnych pakietach, które można wielokrotnie wykorzystywać. Dzięki nim możesz przestać myśleć o kodzie szczegółowo, a zacząć patrzeć na niego w szerszej perspektywie.

Sortowanie i zagnieżdżona iteracja

4 Część II

Porządkowanie danych

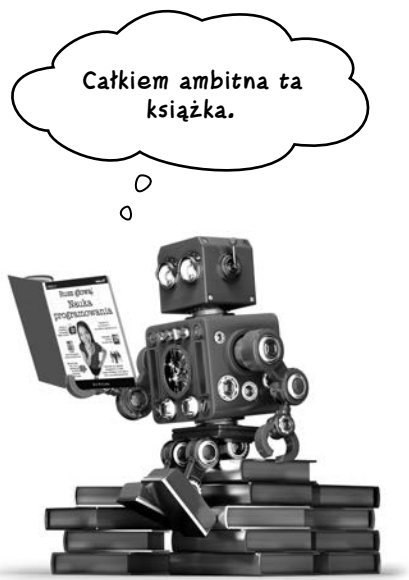
Czasami domyślna kolejność danych jest po prostu niezadowalająca. Możesz na przykład mieć listę najlepszych wyników z automatów z lat 80., ale chciałbyś ją uporządkować alfabetycznie według nazwy gry. Możesz mieć listę, na której zapisujesz, ile razy Twoi „koledzy" z pracy podłożyli Ci świnię — miło byłoby wiedzieć, który z nich przoduje w tej dziedzinie. W tym celu będziesz się jednak musiał nauczyć sortowania danych, a do tego z kolei będzie konieczne, abyś zaznajomił się z nieco bardziej rozbudowanymi algorytmami od tych, z którymi miałeś dotąd styczność. Będziesz też musiał dowiedzieć się, jak działają zagnieżdżone pętle, a także zacząć brać pod uwagę wydajność kodu, jaki piszesz. Dalej, rozwijaj swoje myślenie komputacyjne!

6

Tekst, łańcuchy i heurystyki

Składanie wszystkiego w całość

Masz już całkiem pokaźne supermoce. Nadszedł czas, abyś je wykorzystał. W tym rozdziale zintegrujemy ogół zebranej dotąd wiedzy, tworząc przy tym **coraz bardziej imponujący kod**. Będziesz miał ponadto okazję poszerzyć swoją wiedzę i rozwinąć swoje umiejętności kodowania. Mówiąc konkretniej, w tym rozdziale zbadamy, jak pisać kod, który pobiera tekst, dzieli go na części, a następnie przeprowadza na nim **analizę danych**. Dowiemy się też, czym jest **heurystyka**, i zaimplementujemy przykład takowej. Przygotuj się na konkretny rozdział o prawdziwym kodowaniu!



7

Moduły, metody, klasy i obiekty

Myślenie modularne

Twój kod nabiera rozmiarów i złożoności. W związku z tym coraz potrzebniejsze stają się sposoby na abstrahowanie, modularyzację i organizowanie kodu. Widziałeś już, że funkcji można używać do grupowania wierszy kodu w pakiety nadające się do wielokrotnego użytku. Zetknąłeś się też ze zbiorami funkcji i zmiennych, które można umieszczać w modułach, tak aby łatwiej było się nimi dzielić i z nich korzystać. W tym rozdziale wrócimy do tematu modułów i dowiemy się, jak korzystać z nich jeszcze sprawniej (tak abyś mógł bezproblemowo dzielić się swoim kodem z innymi), po czym przyjrzymy się najważniejszym elementom związanym z wielokrotnym stosowaniem kodu: *obiektom*. Przekonasz się, że obiekty Pythona są wszechobecne i gotowe do wykorzystania.

Rekurencja i słowniki

8 Poza iteracją i indeksami

Nadszedł czas, abyś wkroczył ze swoim myśleniem obliczeniowym na wyższy poziom. W tym rozdziale zajmiemy się właśnie tym. Do tej pory radośnie sobie kodowaliśmy w stylu iteracyjnym — tworzyliśmy struktury danych, takie jak listy, łańcuchy i zakresy liczb, oraz pisaliśmy kod wykonujący obliczenia na bazie iteracji. W tym rozdziale spojrzymy na to wszystko inaczej, najpierw pod względem obliczeń, a potem z perspektywy struktur danych. Przyjrzymy się stylowi obliczania opartego na rekurencji, czyli wykorzystaniu kodu wywołującego samego siebie. Poszerzymy zakres struktur danych, na których możemy pracować — przyjrzymy się słownikom, które są raczej *tablicami asocjacyjnymi* niż listami. Później wykorzystamy te struktury danych do narobienia niezłego bałaganu. Muszę Cię od razu przestrzec, że potrzeba czasu na przyswojenie sobie tematyki poruszonej w tym rozdziale, ale włożony w to wysiłek zwróci Ci się z nawiązką.

Zapisywanie i zwracanie plików

9 Trwałość danych

Wiesz już, że można zapisywać wartości w zmiennych, ale po wykonaniu programu — puf! — tracisz je na zawsze. To właśnie w tym miejscu do gry wkracza *trwałe* magazynowanie danych, umożliwiające nieco dłuższe przechowywanie wartości i danych. Większość urządzeń, na których będziesz używać Pythona, również korzysta z pamięci trwałej w rodzaju dysków twardych i pamięci flash bądź ma też dostęp do chmury. W tym rozdziale dowiesz się, jak pisać kod, który pozwala na zapisywanie danych w plikach i ich zwracanie. Do czego może Ci się to przydać? Między innymi do zapisywania konfiguracji użytkowników, przechowywania wyników obszernych analiz dla Twojego szefa, wczytywania obrazu do kodu w celu jego przetworzenia, pisania kodu do przeszukiwania wszystkich e-maili z okresu ostatnich dziesięciu lat, reformatowania danych do wykorzystania w arkuszu kalkulacyjnym — długo można by tak wymieniać, ale chyba lepiej od razu przejść do rzeczy.

10 Używanie webowych API

Żądania i odpowiedzi

Potrafisz już napisać świetny kod, więc powinieneś zacząć się komunikować. W internecie czeka na Ciebie ogrom danych. Potrzebujesz danych meteorologicznych? Chcesz skorzystać z olbrzymiej bazy danych przepisów kulinarnych? Czy może interesują Cię wyniki sportowe? A może chciałbyś się pobawić muzyczną bazą danych z informacjami o wykonawcach, albumach i utworach? Po to wszystko możesz sięgnąć za pomocą **webowych interfejsów API**. Aby móc z nich skorzystać, będziesz musiał się dowiedzieć nieco więcej o tym, jak działa internet, jak posługiwać się terminologią sieci i jak używać kilku nowych modułów Pythona: `requests` i `json`. W tym rozdziale zapoznamy się z webowymi API i pomożemy Ci się wznieść na wyższy poziom — w zasadzie polecimy z nimi w kosmos.

Widżety, zdarzenia i zachowania emergentne

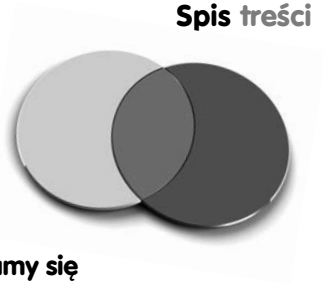
11 Interaktywność

Napisałeś już wprawdzie parę aplikacji graficznych, ale nie utworzyłeś jeszcze prawdziwego interfejsu użytkownika. Innymi słowy, nie napisałeś jeszcze programu, z którego użytkownik mógłby korzystać za pośrednictwem graficznego interfejsu użytkownika (ang. *graphical user interface* — GUI). W tym celu będziesz musiał zacząć myśleć o działaniu programu w inny — **reaktywny** — sposób. Kiedy użytkownik na przykład naciska klawisz, kod powinien wiedzieć, jak zareagować i co zrobić dalej. Kodowanie interfejsów znacznie różni się od standardowego kodowania proceduralnego i wymaga innego podejścia do myślenia o problemach. W tym rozdziale napiszesz swój pierwszy prawdziwy GUI i nie, nie będzie to prosta lista zadań do wykonania ani kalkulator BMI, tylko coś o wiele ciekawszego. Napiszemy symulator sztucznego życia z emergentymi zachowaniami. Co to właściwie znaczy? Tego dowiesz się na następnej stronie.

Programowanie obiektowe

12 Wycieczka do Obiektowa

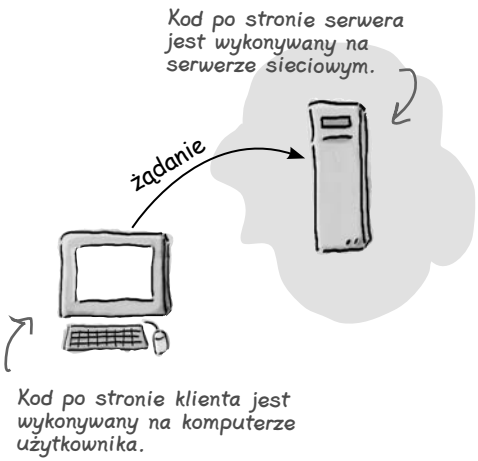
W tej książce posługiwałeś się funkcjami do abstrahowania kodu. Kodowaniem zajmowałeś się **proceduralnie**, czyli używając prostych instrukcji, instrukcji warunkowych i pętli `for/while` z wykorzystaniem funkcji, co trudno uznać w ścisłym sensie za **programowanie obiektowe**. Właściwie to *w ogóle* nie jest programowanie zorientowane obiektowo! Przyjrzeliśmy się obiektom i omówiliśmy, jak używa się ich w kodzie, ale sam jeszcze nie stworzyłeś ani jednego własnego obiektu, ani nie próbowałeś tworzyć kodu zorientowanego obiektowo. Nadszedł zatem czas, abyś opuścił tę zapadłą, proceduralną dziurę, w której byłeś do tej pory uwięziony. W tym rozdziale dowiesz się, dlaczego korzystanie z obiektów znacząco umili Ci życie — przynajmniej w **sensie programistycznym** (niestety nie starczy w tej książce miejsca, żebyśmy mogli Ci pomóc w innych dziedzinach Twojego życia). Musimy Cię tylko ostrzec: po nauczeniu się programowania obiektowego nie będziesz już chciał programować w żaden inny sposób. Bądź więc tak miły i wyślij nam chociaż pocztówkę, kiedy już zdołasz zwiedzić Obiektowo.



Dodatek. Coś na deser

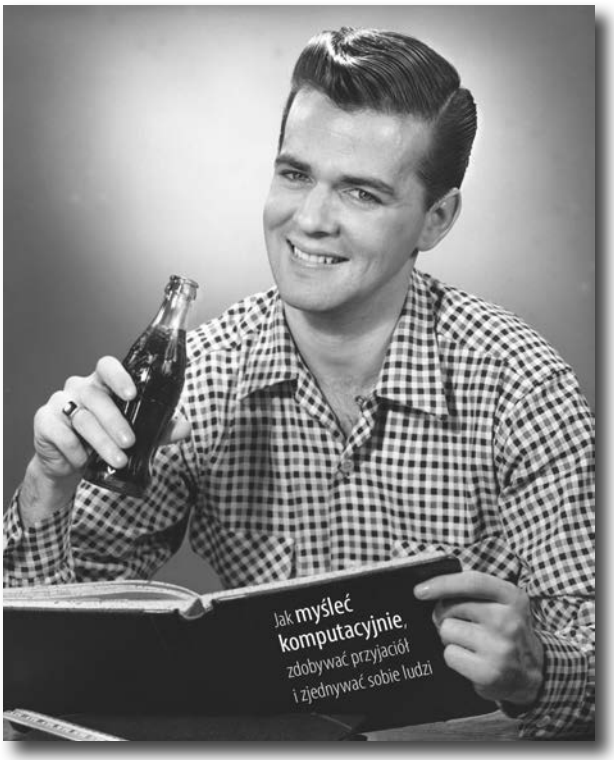
A Dziesięć najważniejszych tematów (których nie omówiliśmy)

Zdołaliśmy omówić wiele obszarów tematycznych i pomału zbliżamy się do końca. Smutno tak się rozstawać, ale nie wybaczyłbym sobie, gdybym wysłał Cię w szeroki świat bez odrobiny przygotowania. Niestety nie zmieszczę w tym stosunkowo krótkim rozdziale wszystkiego, co musisz wiedzieć. Właściwie już *spróbowałem* tutaj zmieścić to wszystko, co powinieneś wiedzieć o programowaniu w Pythonie (a o czym nie wspomniałem w pozostałych rozdziałach), ale musiałem zmniejszyć wielkość pisma do 0,00004 punktu, więc nikt nie potrafił tego przeczytać. Postanowiłem więc wyciąć cały ten tekst i wspomnieć w tym dodatku tylko o najciekawszych wątkach. Dotarłeś niniejszym do *samego końca* książki. Może poza indeksem (który jest niesamowitą lekturą).



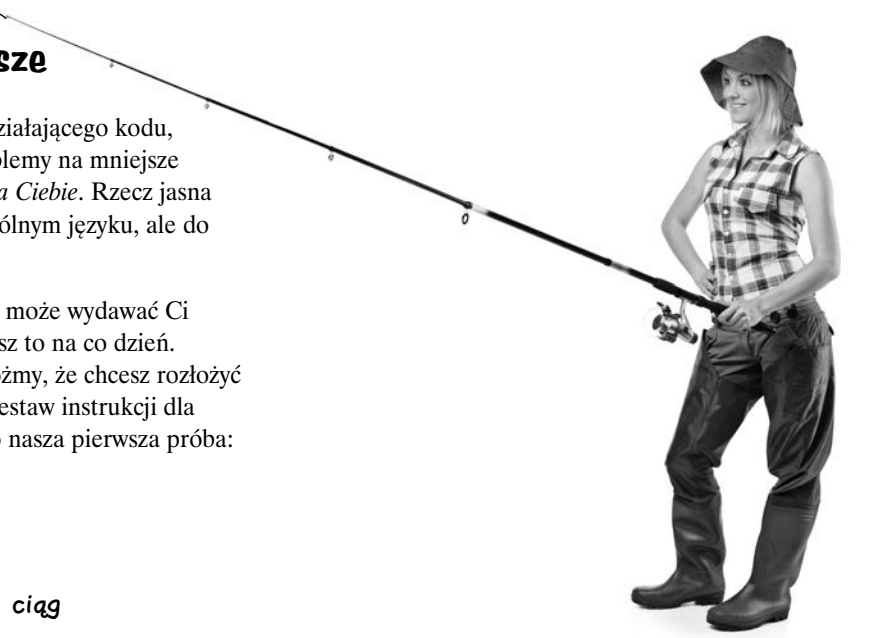
1. Myślenie komputacyjne

Zaczynamy



Umiejętność myślenia komputacyjnego daje Ci kontrolę. Jest tajemnicą poliszynela, że otaczający nas świat staje się coraz silniej połączony, konfigurowalny, programowalny i — co tu dużo mówić — **komputacyjny**. Możesz wobec tego albo pozostać biernym uczestnikiem tej rzeczywistości, albo *nauczyć się kodowania*. Umiejętność kodowania pozwoli Ci zostać twórcą czy też dowódcą — będziesz mógł powiedzieć wszystkim tym komputerom, co mają zrobić *dla Ciebie*. Kiedy umiesz kodować, masz kontrolę nad swoim losem (a już na pewno jesteś w stanie zaprogramować swój połączony z internetem system zraszaczy w ogródku). Jak się jednak nauczyć kodować? Przede wszystkim musisz nauczyć się **myśleć komputacyjnie**. Kolejnym krokiem jest zapoznanie się z **językiem programowania**, który pomoże Ci się dogadać ze swoim komputerem, telefonem lub dowolnym innym urządzeniem z procesorem w środku. Co zyskasz dzięki temu? Czas, władzę i więcej okazji do zajęcia się tymi wszystkimi twórczymi przedsięwzięciami, na których Ci zależy. Zacznijmy zatem...

Dzielenie zadań na mniejsze



Zanim napiszesz swój pierwszy fragment działającego kodu, będziesz musiał nauczyć się rozkładać problemy na mniejsze działania, które komputer może *wykonać za Ciebie*. Rzecz jasna musisz też porozumiewać się z nim we wspólnym języku, ale do tego tematu jeszcze wrócimy.

Rozkładanie problemów na mniejsze kroki może wydawać Ci się czymś nowym, ale w rzeczywistości robisz to na co dzień. Przyjrzyjmy się prostemu przykładowi: załóżmy, że chcesz rozłożyć zajęcie, jakim jest łowienie ryb, na prosty zestaw instrukcji dla robota, który zajmie się tym za Ciebie. Oto nasza pierwsza próba:

Roztóżmy proces łowienia ryb na ciąg prostych i zrozumiałych kroków.

Poniższe kroki wykonujemy po kolei.

1. Załóż robaka na haczyk.
2. Zarzuć przynętę do wody.
3. Przyglądaj się spławikowi, dopóki się nie zanurzy.
4. Wyciągnij rybę.
5. Jeśli skończyłeś wędkować, idź do domu; w przeciwnym razie wróć do punktu 1.

Niektóre kroki są prostymi wyrażeniami — instrukcjami — takimi jak „zarzuć przynętę do wody".

Wykonanie niektórych instrukcji musi być poprzedzone spełnieniem określonego warunku.

Ta instrukcja zostaje wykonana dopiero po zanurzeniu się spławika z poprzedniej instrukcji.

Instrukcje mogą także uwzględniać podejmowanie decyzji, na przykład dotyczących powrotu do domu lub dalszego wędkowania.

Zauważ, że instrukcje często zawierają powtarzające się elementy, tak jak tutaj: jeśli nie idziesz do domu, to zamiast tego wracasz do początku i ponownie wykonujesz instrukcje, aby złowić kolejną rybę.

Powyższe instrukcje możesz potraktować jako **przepis** na łowienie ryb. Jak to bywa z przepisami, znajdziesz tu ciąg kroków, które wykonane we właściwej kolejności pozwalają na uzyskanie określonego rezultatu (w tym przykładzie — złowienie ryb).

Zauważ, że większość kroków składa się z prostych instrukcji, takich jak „zarzuć przynętę do wody" lub „wyciągnij rybę". Niektóre są jednak inne, ponieważ bazują na warunku takim jak „czy spławik jest na wodzie, czy pod wodą?". Instrukcje mogą także kierować przepływem przepisu, tak jak w wypadku „jeśli nie skończyłeś wędkować, wróć do początku i załóż kolejnego robaka na haczyk", lub warunku decydującego o zakończeniu, czyli „jeśli skończyłeś, idź do domu".

Przekonasz się, że takie proste wyrażenia, czy też instrukcje, leżą u podstaw kodu. W rzeczy samej każda aplikacja bądź każdy program, z jakiego kiedykolwiek korzystałeś, był jedynie (być może dość dużym) zbiorem prostych instrukcji, mówiących komputerowi, co ma zrobić.

Ćwiczenie

Prawdziwe przepisy nie uwzględniają jedynie tego, co należy zrobić, ale także wymieniają przedmioty potrzebne do przyrządzenia określonego dania, w tym między innymi miarki, narzędzia, roboty kuchenne, oraz oczywiście składniki. Jakie przedmioty znalazłyby się w przepisie na łowienie ryb? Zakreśl wszystkie takie przedmioty w przepisie z poprzedniej strony i od razu sprawdź na końcu tego rozdziału, czy wszystko uwzględniłeś.

Ta książka jest jednocześnie zeszytem ćwiczeń, w którym możesz swobodnie pisać i kreślić.

Zaostrz ołówek

W pierwszej kolejności musisz zrozumieć, że komputery robią dokładnie to, co im mówisz — ani mniej, ani więcej. Przyjrzyj się naszemu przepisowi na łowienie ryb z poprzedniej strony. Gdybyś był robotem i postąpił dokładnie według tych instrukcji, jakie problemy mógłbyś napotkać? Czy sądzisz, że ten przepis naprawdę pozwoliłby Ci złowić rybę?

- ☐ A. Jeżeli w wodzie nie ma ryb, to będziesz zajmował się wędkowaniem przez bardzo długi czas (mniej więcej całą wieczność).
- ☐ B. Nie weźmiesz pod uwagę tego, że robak mógłby zlecieć z haczyka i że będziesz musiał założyć nowego.
- ☐ C. Co się stanie, jeśli robaki się skończą?
- ☐ D. Czy wskazaliśmy, co zrobić z rybą po wyciągnięciu jej z wody?
- ☐ E. W tym przepisie nie ma ani jednego słowa o wędce.
- ☐ F. _____

_____

_____

_____

_____

Czy przychodzą Ci do głowy jeszcze jakieś problemy?

Jeśli nie jesteś fanatykiem wędkarstwa: to coś to spławik, zwany też pływakiem bądź pławikiem. Kiedy ryba bierze, spławik zanurza się w wodzie.

Odpowiedzi do ćwiczeń z serii „Zaostrz ołówek" znajdują się na końcu rozdziałów.

Przepis jest jednak dobrą analogią do zestawu instrukcji używanych przez komputery. Z tym terminem zetkniesz się jeszcze wielokrotnie, zarówno w tym podręczniku, jak i w bardziej zaawansowanych książkach o programowaniu. Co więcej, niejedna książka o powszechnie stosowanych technikach wytwarzania oprogramowania nazywa się książką kucharską czy zbiorem przepisów. Jeśli jednak zależy Ci na języku technicznym, to proszę bardzo: informatycy i poważni programiści nazywają przepisy **algorytmami**. Czym jest algorytm? Właściwie to przepisem — sekwencją instrukcji, umożliwiającą rozwiązanie jakiegoś problemu. Algorytmy często zapisuje się przy użyciu nieformalnego kodu, zwanego **pseudokodem**.

Już wkrótce wrócimy do tematu pseudokodu...

Warto pamiętać, że niezależnie od tego, czy ma się do czynienia z przepisem, pseudokodem czy algorytmem, chodzi przede wszystkim o to, aby wypracować wysokopoziomowy opis tego, jak można rozwiązać dany problem, jeszcze zanim przejdzie się do szczegółów związanych z pisaniem kodu, który komputer może zrozumieć i wykonać.

Jak się przekonasz, samo kodowanie jest dzięki temu prostsze i mniej podatne na błędy.

W tej książce będziemy posługiwać się wszystkimi tymi terminami zamiennie. Pamiętaj, żeby w swojej kolejnej rozmowie o pracę użyć pojęcia *algorytm* lub *pseudokod*, a zaimponujesz pracodawcy (choć mimo to *przepis* wciąż jest całkowicie dobrym terminem).

Podobnie jak każdą potrawę można przyrządzić według więcej niż jednego przepisu, określony problem można rozwiązać przy użyciu różnych algorytmów. Rzecz w tym, że niektóre z nich są po prostu „smaczniejsze".

Magnesiki kodu

Poćwiczmy trochę układanie przepisów algorytmów. Na drzwiach lodówki ułożyliśmy z magnesów algorytm od babci, pozwalający na przyrządzenie omletu z trzech jaj, ale ktoś postanowił zabawić się naszym kosztem i wszystko pomieszał. Czy możesz ułożyć magnesiki we właściwej kolejności, tak aby algorytm zadziałał? Miej na uwadze, że babcia robi dwa rodzaje omletów: zwykły i z serem. **Koniecznie sprawdź poprawną odpowiedź na końcu rozdziału.**

Utóż tutaj magnesiki tak, aby algorytm zadziałał.

Jeśli chcesz omlet z serem:

Posyp serem

Dopóki żółtko nie jest w pełni zmieszane z białkiem:

Przełóż jajka na patelnię

Dopóki jajka nie są w pełni ścięte:

Zdejmij patelnię z ognia

Mieszaj jajka

Podaj

Podgrzej patelnię

Ubij jajka

Rozbij trzy jajka do miski

Przelej jajka na patelnię

Jak działa kodowanie?

Masz zadanie, które chciałbyś, aby komputer wykonał za Ciebie. Wiesz też, że musisz to zadanie rozłożyć na ciąg zrozumiałych dla komputera instrukcji, ale *jak właściwie* możesz kazać mu coś zrobić? Tutaj właśnie do gry wchodzi język programowania, który umożliwia Ci opisanie zadania w sposób zrozumiały *zarówno* dla komputera, *jak i* dla Ciebie samego. Zanim jednak zgłębimy temat języków programowania, przyjrzyjmy się krokom związanym z tworzeniem kodu:

1 Sformułuj algorytm

Na tym etapie przekształcasz problem do rozwiązania lub zadanie do wykonania w wysokopoziomowy przepis, fragment pseudokodu lub algorytm opisujący kroki, które należy wykonać, aby komputer uzyskał pożądany rezultat.

1. Załóż robaka na haczyk.
2. Zarzuć przynętę do wody.
3. Przyglądaj się spławikowi, dopóki się nie zanurzy.
4. Wyciągnij rybę.
5. Jeśli skończyłeś wędkować, idź do domu; w przeciwnym razie wróć do punktu 1.

Rozpisujemy swoje rozwiązanie, zanim jeszcze weźmiemy się za przetłumaczenie go na język programowania.

2 Napisz program

Gotowy przepis przekładasz na zbiór konkretnych instrukcji, napisanych w języku programowania. Jest to etap kodowania, którego rezultat nazywamy programem lub po prostu „Twoim kodem" (lub kodem źródłowym, jeśli zależy nam na bardziej formalnym języku).

```
def ryba_na_haczyku():
    print('Złapałem rybę! ')

def czekaj():
    print('Czekam… ')

print('Biorę robaka')
print('Zakładam robaka na haczyk')
print('Zarzucam przynętę')

while True:
    odpowiedz = input('Czy spławik jest zanurzony pod wodą? ')
    if odpowiedz == 'tak':
        rusza_sie = True
        print('Złapała haczyk!')
        ryba_na_haczyku()
    else:
        czekaj()
```

Na etapie „kodowania" przekształcasz swój algorytm w kod (co jest skrótowym określeniem kodu źródłowego), który będzie można wykonać w kolejnym kroku.

3 Uruchom program

Na koniec podajesz kod źródłowy komputerowi, który zaczyna wykonywać Twoje instrukcje. W zależności od języka, jakim się posługujesz, proces ten może się nazywać *interpretowaniem, uruchomieniem, ewaluacją* lub *wykonaniem kodu.*

Tych terminów też będziemy używać zamiennie.

Po napisaniu kodu źródłowego możesz go wykonać. Jeśli wszystko pójdzie dobrze, a kod został odpowiednio utożony, komputer zwróci Ci pożądany rezultat.

Ale po jakiemu to?

Język programowania powinieneś traktować jako język szczególnego zastosowania, stworzony konkretnie z myślą o formułowaniu zadań do wykonania przez komputer. Języki programowania umożliwiają Ci opisywanie Twoich przepisów w sposób wystarczająco jasny i dokładny, aby komputer mógł je zrozumieć.

W nauce języka programowania musisz zwrócić uwagę na dwie rzeczy: co można takim językiem powiedzieć oraz co to oznacza. Informatycy nazywają to **składnią** i **semantyką** języka. Na razie zapamiętaj, że te pojęcia istnieją; z czasem będziesz dowiadywał się o nich coraz więcej.

Podobnie jak języków naturalnych, języków programowania jest wiele, a — jak już być może się domyśliłeś — w tej książce będziemy posługiwać się Pythonem. Przybliżmy sobie teraz nieco kwestię Pythona i języków programowania w ogóle…

Spokojnie

Nie martw się. W tej chwili nikt nie oczekuje od Ciebie umiejętności czytania i pisania kodu. Jesteś na samym początku książki — na razie dopiero zaznajamiasz się z kodem, tym, jak wygląda i jak działa. Teraz chodzi tylko o to, abyś sobie przyswoił przedstawione tutaj informacje.

Sam zobaczysz, że omówione w tej książce techniki będziesz mógł odnieść niemalże do dowolnego innego języka programowania, z jakim zetkniesz się w przyszłości.

POMIDOR, POMIDOR

Po lewej widnieją instrukcje napisane po polsku, a po prawej ich odpowiedniki w języku programowania. Narysuj linie łączące obydwie wersje instrukcji. Pierwszą sam narysowałem. Zanim przejdziesz dalej, koniecznie sprawdź poprawne odpowiedzi na końcu rozdziału.

Napisz „Cześć!” na ekranie.

Jeśli temperatura jest wyższa niż 22 stopnie, napisz na ekranie „Włóż krótkie spodnie”.

Lista zakupów z chlebem, mlekiem i jajkami.

Nalej pięć szklanek napoju.

Zadaj użytkownikowi pytanie „Jak masz na imię?”.

```
for liczba in range(0, 5):
    nalej_szklanke()
```

```
imie = input('Jak masz na imię? ')
```

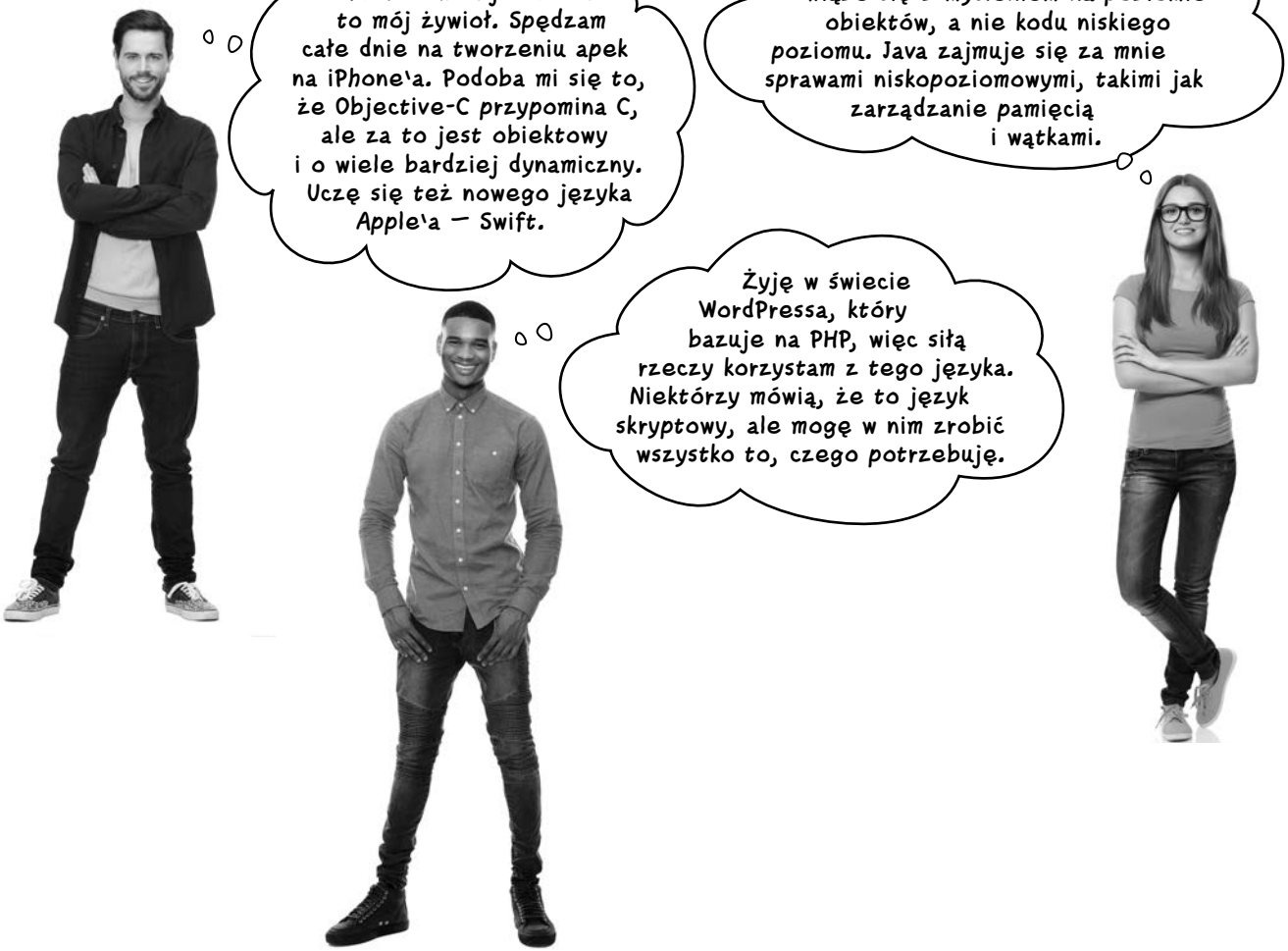
```
if temperatura > 22:
    print('Włóż krótkie spodnie')
```

```
lista_zakupow = ['chleb', 'mleko', 'jajka']
```

```
print('Cześć!')
```

Świat języków programowania

Jeśli czytasz tę książkę, być może słyszałeś już o różnych językach programowania. Wystarczy zajrzeć do działu informatycznego księgarni, żeby zetknąć się z językami takimi jak Java, C, C++, LISP, Scheme, Objective-C, Perl, PHP, Swift, Clojure, Haskell, COBOL, Ruby, Fortran, Smalltalk, BASIC, Algol, JavaScript i oczywiście Python, że wymienimy zaledwie garstkę. Być może zastanawiasz się, skąd wzięły się te wszystkie nazwy. Prawda jest taka, że z nazwami języków programowania jest jak z nazwami zespołów rockowych — zazwyczaj mają one sens głównie dla swoich twórców. Weźmy na przykład Javę, która wzięła swoją nazwę od kawy, bo pierwsza nazwa, Oak (dąb), okazała się zajęta. Haskell nosi nazwę po matematyku, a C nazywa się tak, a nie inaczej, ponieważ jest następcą języków A i B z Bell Labs. Dlaczego jednak istnieje tyle języków i do czego one służą? Posłuchajmy, co parę osób ma do powiedzenia na temat języków, z których korzysta:



Używam głównie języka C, bo piszę elementy systemów operacyjnych, które muszą być superwydajne. Dla mnie liczy się każdy cykl procesora i każda alokacja pamięci.

Może zabrzmi to przesadnie akademicko, ale uwielbiam języki w rodzaju Scheme i Lisp. Zależy mi na funkcjach wyższego rzędu i abstrakcjach. Cieszę się, że języki funkcyjne takie jak Clojure znajdują realne zastosowanie w branży.

Jestem programistką stron internetowych, więc korzystam głównie z JavaScriptu. To nim de facto posługują się wszystkie przeglądarki, a poza tym można w nim także tworzyć usługi backendowe.

Jestem administratorem systemu i często piszę skrypty systemowe w Perlu. To dość lapidarny, ale zarazem ekspresyjny język. Wystarczy odrobina kodu, żeby zrobić bardzo dużo.

Tyle możliwości...

Jak widać, języków i opinii na ich temat jest multum, a to jedynie wierzchołek góry lodowej. Być może zauważyłeś też, że z różnymi językami wiąże się różna terminologia. Z czasem te wszystkie terminy będą stawać się dla Ciebie coraz bardziej zrozumiałe. Na razie jednak pamiętaj, że istnieje duży wybór języków, a każdego dnia powstają coraz to nowsze.

Z czego zatem powinniśmy skorzystać w tej książce? Przede wszystkim zależy nam na nauce myślenia *komputacyjnego*. W ten sposób bez względu na to, z jakimi językami zetkniesz się w przyszłości, będziesz mógł się ich z łatwością nauczyć. Niemniej jednak od *jakiegoś* języka musimy zacząć. Jak już wiesz, będzie to Python. Dlaczego? Nasi koledzy dobrze to powyżej uzasadnili: Python jest uznawany za jeden z najlepszych języków dla początkujących, ponieważ jest niezwykle czytelny i spójny. Jest też bardzo skuteczny, ponieważ niezależnie od tego, co chcesz w nim zrobić (zarówno w ramach nauki z pomocą tej książki, jak i poza tym), możesz skorzystać z różnych rozszerzeń (które nazywamy *modułami* lub *bibliotekami*) oraz poprosić społeczność programistów o pomoc. Co więcej, niektórzy programiści są zdania, że praca z Pythonem daje dużo więcej *zabawy* niż z innymi językami. Jest to zatem idealny kandydat.

Zaostrz ołówek

Spójrz, jak łatwo pisać w Pythonie

Nie znasz jeszcze Pythona, ale pewnie potrafisz odgadnąć, jak działa napisany w nim kod. Przyjrzyj się poszczególnym linijkom kodu poniżej i spróbuj zgadnąć, co która robi. Zapisz swoje odpowiedzi obok. Jeżeli nie masz pomysłu, odpowiedzi znajdziesz na następnej stronie. Pierwsze pole już wypełniliśmy za Ciebie.

```
klienci = ['Julian', 'Ada', 'Daniel', 'Sylwia']

zwyciezca = random.choice(klienci)

smak = 'wanilia'

print('Gratulacje, ' + zwyciezca + '! Wygrałaś/-eś
pucharek lodów!')

pytanie = 'Czy chcesz wisienkę na wierzchu? '

wisienka = input(pytanie)

zamowienie = ' pucharek lodów o smaku ' + smak

if (wisienka == 'tak'):
   zamowienie = zamowienie + ' z wisienką na
wierzchu'

print(zwyciezca + ' dostanie jeden ' + zamowienie +
' już za momencik...')
```

Tworzy listę klientów.

Python Output

```
Gratulacje, Sylwia! Wygrałaś/-eś pucharek lodów!
Czy chcesz wisienkę na wierzchu? tak
Sylwia dostanie jeden pucharek lodów o smaku wanilia
z wisienką na wierzchu już za momencik...
```

Jako ułatwienie masz tutaj przegląd tego, co kod zwraca. Czy sądzisz, że powyższy kod za każdym razem zwraca to samo?

Zaostrz ołówek. Rozwiązanie

Spójrz, jak łatwo pisać w Pythonie

Nie znasz jeszcze Pythona, ale pewnie potrafisz odgadnąć, jak działa napisany w nim kod. Przyjrzyj się poszczególnym linijkom kodu poniżej i spróbuj zgadnąć, co która robi. Zapisz swoje odpowiedzi obok. Jeżeli nie masz pomysłu, odpowiedzi znajdziesz na następnej stronie. Pierwsze pole już wypełniliśmy za Ciebie.

`klienci = ['Julian', 'Ada', 'Daniel', 'Sylwia']`	*Tworzy listę klientów.*
`zwyciezca = random.choice(klienci)`	*Losowo wybiera jednego z klientów.*
`smak = 'wanilia'`	*Przydziela zmiennej o nazwie smak wartość tekstową 'wanilia'.*
`print('Gratulacje, ' + zwyciezca + '! Wygrałaś/-eś pucharek lodów!')`	*Wyświetla w konsoli gratulacje z imieniem zwycięzcy. Jeżeli na przykład wygrała Ada, ta linijka kodu wyświetla tekst 'Gratulacje, Ada! Wygrałaś/-eś pucharek lodów!'.*
`pytanie = 'Czy chcesz wisienkę na wierzchu? '`	*Przydziela zmiennej o nazwie pytanie wartość tekstową 'Czy chcesz wisienkę na wierzchu? '.*
`wisienka = input(pytanie)`	*Prosi użytkownika o wpisanie tekstu, który następnie przypisuje zmiennej wisienka. Zauważ, że w konsoli Pythona w pierwszej kolejności pojawia się wartość zmiennej pytanie.*
`zamowienie = ' pucharek lodów o smaku ' + smak`	*Nadaje zmiennej zamowienie wartość tekstową ' pucharek lodów o smaku ', po której podaje wartość zmiennej smak.*
`if (wisienka == 'tak'):` `  zamowienie = zamowienie + ' z wisienką na wierzchu'`	*Jeżeli użytkownik odpowiedział „tak" na pytanie „Czy chcesz wisienkę na wierzchu?", do zmiennej zamowienie dodany zostaje tekst ' z wisienką na wierzchu'.*
`print(zwyciezca + ' dostanie jeden ' + zamowienie + ' już za momencik...')`	*Wyświetla w konsoli informację o tym, że zamówienie zwycięzcy losowania za moment do niego trafi.*

Python Output

```
Gratulacje, Sylwia! Wygrałaś/-eś pucharek lodów!
Czy chcesz wisienkę na wierzchu? tak
Sylwia dostanie jeden pucharek lodów o smaku wanilia
z wisienką na wierzchu już za momencik...
```

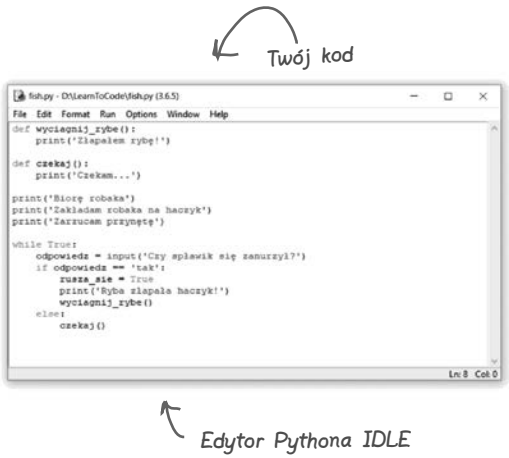
PS. Jeżeli nie możesz się oprzeć pokusie i chcesz wypróbować ten kod, wpisz import random w pierwszej linijce pliku przed jego uruchomieniem. Później dowiesz się, dlaczego tak należy zrobić, ale powinieneś wziąć pod uwagę, że w tej chwili uruchamianie kodu nie jest ani konieczne, ani szczególnie przydatne. Niemniej jednak wiem po prostu, że znajdą się tacy czytelnicy, którzy będą chcieli wszystko zrobić sami. To już leży wyłącznie w Twojej gestii!

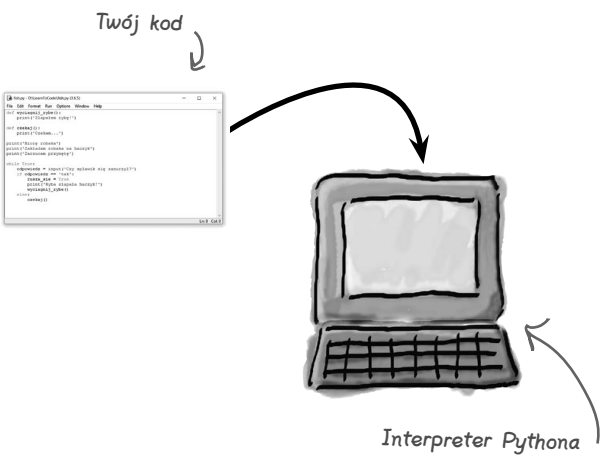
Pisanie i uruchamianie kodu w Pythonie

Skoro już omówiliśmy i przejrzeliśmy fragment kodu, najwyższy czas zastanowić się nad tym, jak napisać i wykonać własny. Jak już wspomniano, może to przebiegać na różne sposoby, w zależności od języka i środowiska. Rozeznajmy się, jak wygląda pisanie i uruchamianie kodu w Pythonie:

1 Pisanie kodu

Na początek musisz napisać kod w edytorze tekstu i zapisać plik. Kod Pythona można napisać w dowolnym edytorze, takim jak Notatnik w Windowsie lub TextEdit w macOS-ie. Większość programistów korzysta jednak ze specjalistycznych edytorów, czyli zintegrowanych środowisk programistycznych (ang. *Integrated Development Environment* — IDE). Dlaczego? IDE przypominają procesory tekstu, ponieważ zapewniają użytkownikowi wiele przydatnych funkcji, takich jak autouzupełnianie najczęściej używanych nazw w Pythonie, podświetlanie elementów składni (i błędów) oraz wbudowane testy. Python ma własne IDE o nazwie IDLE, któremu wkrótce się przyjrzymy.





2 Uruchamianie kodu

Uruchomienie kodu sprowadza się do przekazania go interpreterowi Pythona — programowi, który robi wszystko, co jest konieczne, aby wykonać napisany przez Ciebie kod. Szczegóły tego procesu omówimy już wkrótce, ale na razie wiedz, że z interpretera możesz skorzystać zarówno przez IDLE, jak i bezpośrednio z wiersza poleceń systemu operacyjnego.

3 Interpretacja kodu

O Pythonie pisałem dotąd jako o języku, który jest zrozumiały zarówno dla komputera, jak i Ciebie samego. Wiesz już, że interpreter odczytuje Twój kod i go wykonuje. W tym celu tłumaczy Twój kod na kod maszynowy niższego poziomu, który może zostać wykonany bezpośrednio przez komputer. Nie musisz wiedzieć, jak to się dokładnie odbywa. Zapamiętaj tylko, że interpreter wykonuje każdą instrukcję kodu Pythona.

Nie istnieją głupie pytania

P: Czy nie lepiej byłoby programować komputery w jakimś ludzkim języku? Nie trzeba by się uczyć tych wszystkich specjalistycznych języków programowania.

O: Owszem, byłoby miło. W praktyce jednak języki naturalne są pełne wieloznaczności, co bardzo utrudnia tłumaczenie ich na kod maszynowy. Badacze wprawdzie poczynili pewne postępy w tym kierunku, ale do wykorzystania polskiego, angielskiego czy innego języka naturalnego w programowaniu jest jeszcze daleka droga. Istnieje kilka języków programowania, które mają brzmieć naturalnie, ale okazuje się, że programistom bardziej zależy na przystosowaniu języka do wymogów kodowania niż na jego podobieństwie do języka mówionego.

P: Dlaczego nie używa się po prostu jednego języka programowania?

O: Technicznie rzecz ujmując, wszystkie współczesne języki programowania są równorzędne, ponieważ pozwalają na obliczanie tych samych rzeczy, wobec czego teoretycznie rzeczywiście moglibyśmy używać jednego języka do wszystkiego. Podobnie jednak jak w wypadku języków mówionych, języki programowania mają różną moc ekspresji. Innymi słowy, niektóre czynności programistyczne (na przykład tworzenie witryn internetowych) wykonuje się w jednych językach łatwiej niż w innych. Wybór języka niekiedy też sprowadza się do osobistych preferencji, korzystania z określonej metodyki czy nawet wymogu używania języka stosowanego przez pracodawcę. Jednego możesz być pewien: języki programowania ewoluują i z czasem będzie ich coraz więcej.

P: Czy Python jako język służy przede wszystkim początkującym do zabawy? Czy nauka Pythona pomoże mi, jeśli chcę zostać zawodowym programistą?

O: Python jest poważnym językiem, używanym do tworzenia wielu produktów, które prawdopodobnie znasz i lubisz. Co więcej, Python jest jednym z niewielu profesjonalnych języków uznawanych jednocześnie za znakomity wybór dla początkujących. Dlaczego? W porównaniu z licznymi innymi językami podejście Pythona do wielu kwestii jest proste i spójne (co docenisz z czasem, kiedy już zapoznasz się bliżej zarówno z Pythonem, jak i z innymi językami).

P: Jaka jest różnica między nauką kodowania a myślenia komputacyjnego? To drugie to chyba coś informatycznego?

O: Myślenie komputacyjne jest pewnym podejściem do rozwiązywania problemów, które wyrosło na gruncie informatyki. Myślenie komputacyjne uczy nas rozkładania problemów na czynniki pierwsze, tworzenia algorytmów pozwalających na ich rozwiązywanie oraz uogólniania takich rozwiązań na większą skalę. Często jednak zależy nam na nauczeniu komputera, jak wykonywać takie algorytmy za nas — i tutaj właśnie przechodzimy do kodowania. Kodowanie jest metodą przekazania algorytmu komputerowi (bądź jakiemukolwiek innemu urządzeniu obliczeniowemu, takiemu jak smartfon). Te dwie rzeczy idą zatem ramię w ramię — myślenie komputacyjne pozwala nam tworzyć rozwiązania, które chcemy zakodować, a kodowanie umożliwia przedstawianie tych rozwiązań komputerom. Tak czy inaczej, myślenie komputacyjne bywa wartościowe nawet w sytuacjach niezwiązanych z kodowaniem.

WYSIL SZARE KOMÓRKI

Jak sądzisz, skąd wzięła się nazwa Python? Zaznacz poniżej najbardziej prawdopodobną odpowiedź.

- ☐ **A.** Twórca Pythona uwielbiał węże, a już wcześniej stworzył mniej udany język Cobra.
- ☐ **B.** Twórca był ciekawski, więc zadawał wiele pytań. Stąd Python.
- ☐ **C.** Inspiracją była nazwa pewnej brytyjskiej grupy komediowej.
- ☐ **D.** Python jest skrótem od Programming Your Things, Hosted On the Network (programuj i hostuj swoje rzeczy w sieci).
- ☐ **E.** Inspiracją była nazwa systemu uruchomieniowego Anaconda, na którym Python jest oparty.

C. Monty Python jest brytyjską grupą komediową, znaną z serialu Latający Cyrk Monty Pythona. W ramach pracy domowej obejrzyj sobie parę ich skeczy. Ten typ humoru może Ci mniej lub bardziej odpowiadać, ale społeczność Pythona posługuje się wieloma nawiązaniami do Monty Pythona.

Bardzo krótka historia Pythona

Python 1.0

W holenderskim Centrum Wiskunde & Informatica pojawił się poważny problem: naukowcy mieli trudność z uczeniem się języków programowania. Zgadza się — ówczesne języki programowania wydawały się zbyt skomplikowane i zagmatwane nawet wykształconym, uzdolnionym uczonym. Aby temu zaradzić, Centrum opracowało nowy język z myślą o ułatwieniu nauki, zwany ABC (pewnie sądziłeś, że chodzi o Pythona, co?). Choć ABC odniósł umiarkowany sukces, pewien młody, przedsiębiorczy programista — Guido van Rossum — po weekendowym maratonie oglądania w telewizji powtórek Monty Pythona doszedł do wniosku, że można to wszystko zrobić lepiej. Wykorzystując umiejętności zdobyte dzięki ABC, Guido stworzył Pythona, a dalej już wiadomo, jak się sprawy potoczyły.

Uwaga od redaktora: a właśnie że nie wiadomo! O tym są kolejne akapity.

Python 2.0

Python osiągnął dojrzałość wraz z wersją 2.0, w której pojawił się cały nowy zestaw właściwości mających wspomóc rozrastające się grono programistów tego języka. Ze względu na upowszechnienie się Pythona na całym świecie w wersji 2.0 wprowadzono obsługę znaków wykraczających poza zbiór standardowych liter angielskich. Usprawniono także wiele technicznych aspektów języka, takich jak zarządzanie pamięcią komputerową i obsługa powszechnie spotykanych typów danych, takich jak listy i ciągi znaków.

Zespół programistów Pythona dołożył także starań, aby otworzyć ten język na całą społeczność ludzi, którzy mogliby pomóc go ulepszyć.

No dobrze, o maratonie Monty Pythona akurat zmyślitem.

Python 3.0

Nikt nie jest idealny, wobec czego nadszedł dzień, w którym twórcy Pythona dostrzegli parę aspektów tego języka, które warto by usprawnić. Choć Python znany jest ze swojej prostoty, praktyka wykazała, że pewne jego obszary mogłyby zostać ulepszone, a zarazem pewnych rozwiązań, które nie przetrwały próby czasu, warto by się pozbyć.

Ogół tych zmian sprawił, że niektóre aspekty Pythona 2 przestałyby być obsługiwane. Niemniej jednak twórcy Pythona dopilnowali, aby kod napisany w wersji 2.0 dalej mógł działać. Jeżeli zatem dysponujesz kodem w Pythonie 2, nie przejmuj się — ta wersja wciąż ma się dobrze, ale miej na uwadze, że Python 3 jest przyszłością tego języka.

Latające samochody będą działać na Pythonie. To pewne.

1994 2000 2008 Przyszłość!

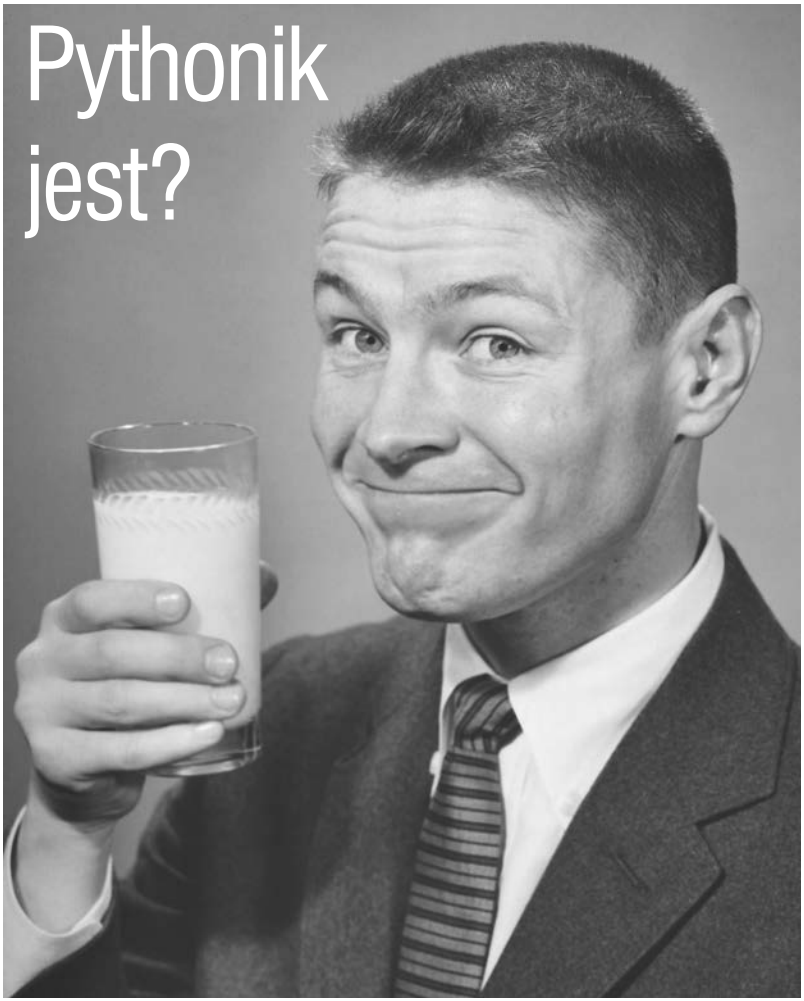
Czyli mamy Pythona 2 i 3. Z której wersji będziemy korzystać i czym się właściwie one różnią?

Dobre pytanie. Owszem, są dwie wersje Pythona. W chwili gdy ta książka była drukowana, aktualnymi wersjami były 3.6 i 2.7.

Określmy teraz, jak będziemy traktować te dwie wersje. Gdybyśmy mogli je gdzieś postawić i spojrzeć na nie z odległości kilometra, odnieślibyśmy wrażenie, że są niezwykle podobne do siebie. Być może nawet nie moglibyśmy dostrzec żadnych różnic. Niemniej jednak takie różnice istnieją i jeśli nie pamiętasz, której wersji używasz, w pewnym momencie możesz się na tym przejechać. Na potrzeby tej książki będziemy korzystać z najnowszej wersji Pythona, czyli 3. Lepiej, żebyś od razu zajął się nauką tej wersji, która to będzie rozwijana w przyszłości.

Tymczasem na świecie jest dużo kodu napisanego w Pythonie 2. Z całą pewnością zetkniesz się z nim w modułach pobranych z sieci, a jeśli zostaniesz programistą, pewnie będziesz musiał zająć się obsługą starego kodu. Po przeczytaniu tej książki będziesz umiał w razie potrzeby dostrzec te wszystkie drobne różnice między Pythonem 3 a 2.

Kiedy piszę o Pythonie 3 i 2, mam na myśli najnowsze wersje (czyli z mojej perspektywy, odpowiednio, 3.6 i 2.7).

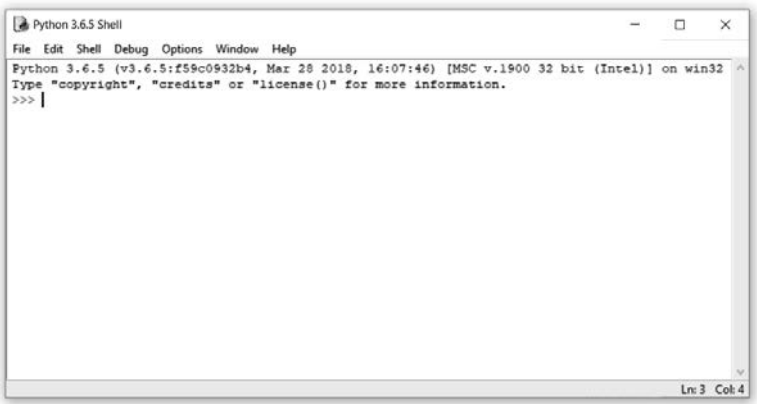


Daleko nie zajdziemy, jeśli jeszcze nie zainstalowałeś Pythona. Jeśli nie poświęciłeś na to czasu wcześniej, zajmij się tym teraz. W tym celu wróć do punktu „Musisz zainstalować Pythona” we wprowadzeniu. Pamiętaj, że jeśli używasz Maca lub pracujesz w Linuksie, to prawdopodobnie masz już zainstalowanego Pythona, z tym że jest to raczej wersja 2, a nie 3. Niezależnie od tego, na jakim systemie pracujesz, konieczne może być zainstalowanie Pythona 3.

Zrób wszystko, co trzeba. Kiedy Python będzie gotowy, będziesz wreszcie mógł się zająć prawdziwym kodem.

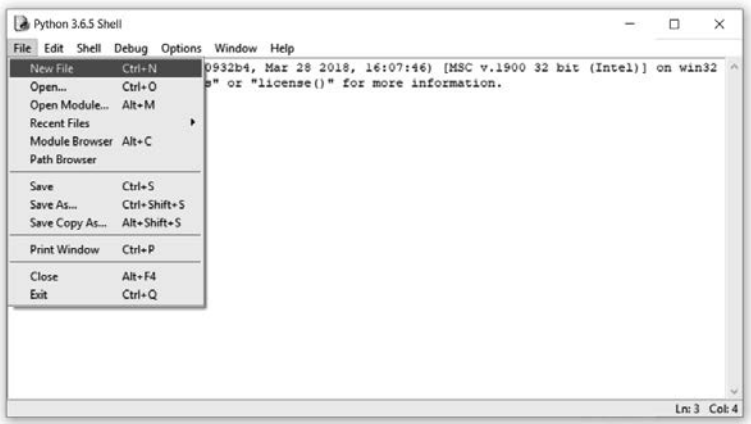
Python — pierwsza próba

Skoro masz już zainstalowanego Pythona, najwyższy czas go wypróbować. Zaczniemy od przygotowania krótkiego programu testowego, aby upewnić się, czy wszystko działa. Przed uruchomieniem programu musisz go najpierw napisać w edytorze. Do tego właśnie posłuży nam IDLE, czyli edytor (czy też zintegrowane środowisko programistyczne) Pythona. Otwórz IDLE, tak jak to już zrobiliśmy we wprowadzeniu. W ramach przypomnienia: na Macu IDLE znajduje się w folderze *Applications/Python 3.x*. W systemie Windows otwórz menu *Start* i wybierz IDLE z menu *Python 3.x*.



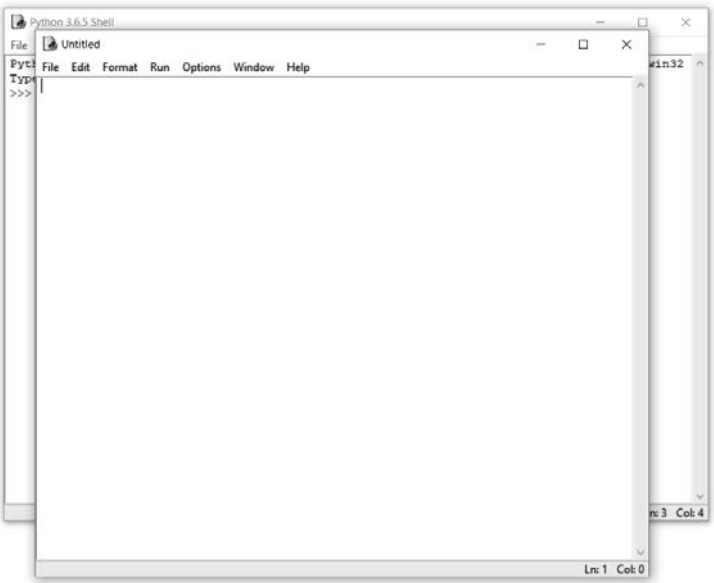
Po uruchomieniu IDLE na ekranie pojawi się interaktywny interpreter, zwany powłoką Pythona. Jeśli jesteś ciekaw, wpisz w nim 1+1 (jeden plus jeden) i naciśnij Enter. Więcej na ten temat przeczytasz w następnym rozdziale.

Po pierwszym uruchomieniu IDLE na ekranie pojawia się interaktywne okno, czyli powłoka Pythona, w której możesz bezpośrednio wpisywać instrukcje Pythona. Możesz także napisać swój kod w edytorze. W tym celu kliknij *File/New File* (plik/nowy plik). Na ekranie pojawi się nowe, puste okno edytora.



Kliknij File/New File, aby otworzyć nowe okno, w którym będziesz mógł wprowadzić kod Pythona.

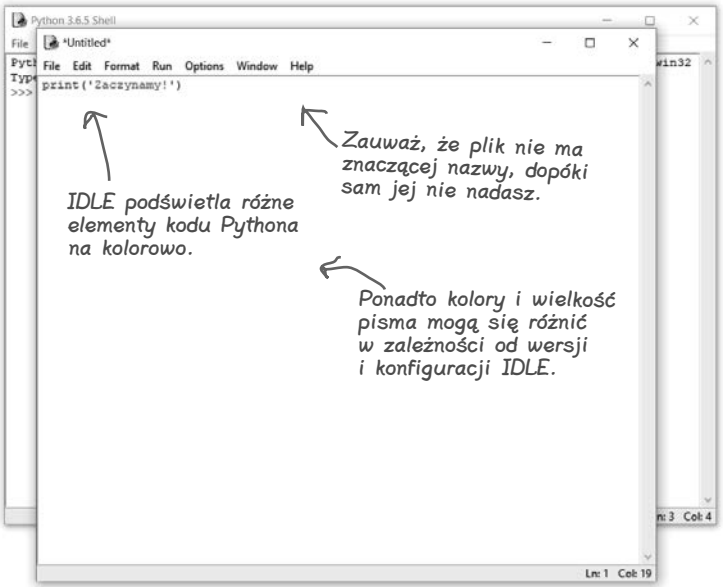
IDLE działa tak jak procesor tekstu, z tym że rozumie kod Pythona i pomaga Ci — podświetla słowa kluczowe Pythona, co ułatwia formatowanie, a niekiedy nawet daje możliwość autouzupełnienia często występujących słów kluczowych.



Po kliknięciu New File powinieneś zobaczyć nowe okno nad oknem powłoki Pythona.

Po utworzeniu nowego pliku i otwarciu nowego, pustego okna edycji wpisz jedną linijkę kodu, która pozwoli Ci przetestować Pythona:

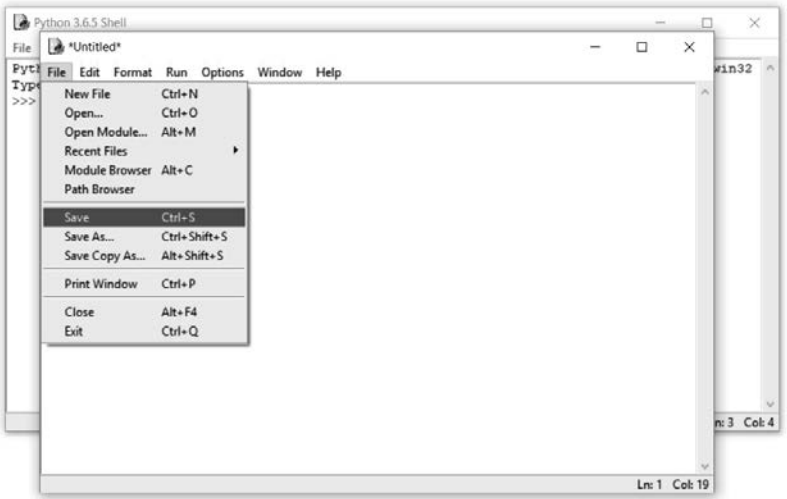
```
print('Zaczynamy!')
```



Zwracaj szczególną uwagę na ortografię i interpunkcję, jako że Python i inne języki programowania nie znoszą błędów.

Zapisywanie wyników swojej pracy

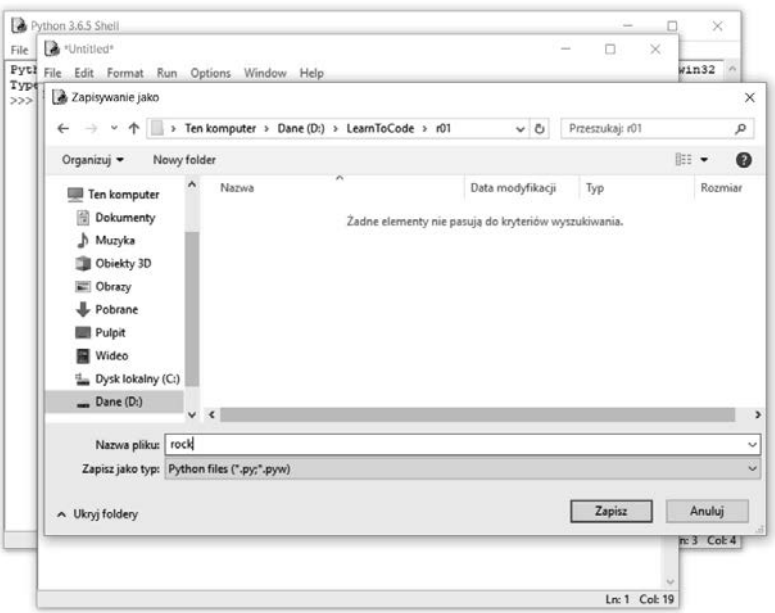
Skoro już wpisałeś pierwszą linijkę kodu, spróbuj ją zapisać.
W tym celu kliknij *File*/*Save* (plik/zapisz):



Musimy zapisać kod przed jego uruchomieniem. W IDLE polecenie Save znajduje się w menu File. Kliknij je i nadaj swojemu programowi nazwę. Kod Pythona zapisujemy w plikach o rozszerzeniu „.py".

Kod źródłowy, plik źródłowy i program są wymiennymi nazwami plików, w których mieści się kod.

Nadaj swojemu kodowi nazwę i rozszerzenie *.py*. Zdecydujmy się na nazwę *rock.py*. Zauważ, że utworzyłem już folder na kod z rozdziału 1. — o nazwie *r01*. Warto, abyś zrobił to samo.



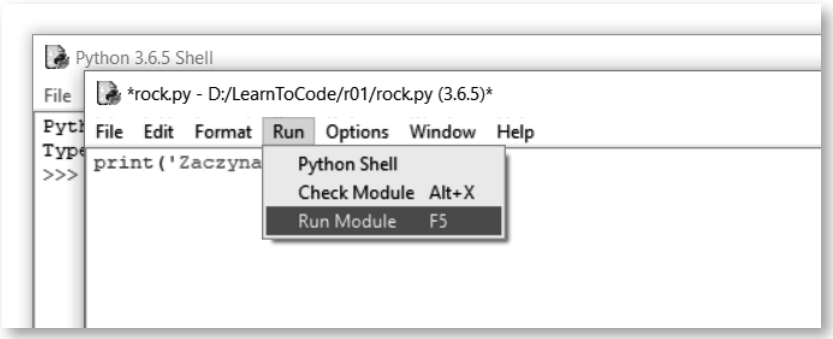
Powinieneś organizować swój kod zgodnie ze strukturą, której sam używam. Jeśli jeszcze tego nie zrobiłeś, rzuć okiem na s. XXXVI we wprowadzeniu.

Po utworzeniu folderu na kod i nadaniu plikowi nazwy kliknij Zapisz.

Jazda próbna

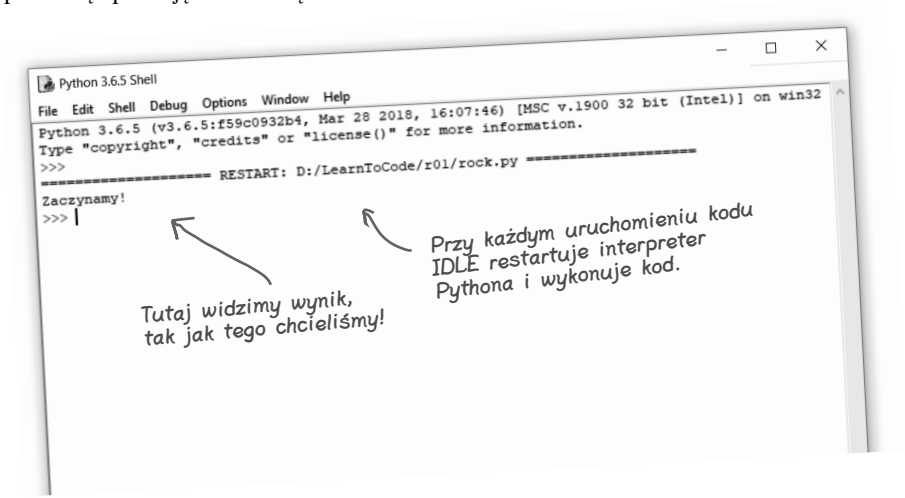
Teraz przechodzimy do rzeczy. Po zapisaniu kodu kliknij polecenie *Run*/*Run Module* (uruchom/uruchom moduł). Spójrz następnie na okno powłoki Pythona, aby zobaczyć wynik programu.

Uruchom swój kod poleceniem Run Module z menu Run. Jeśli nie zapisałeś swojego kodu, zostaniesz poproszony o to przez IDLE.



Gratulacje! Właśnie napisałeś swój pierwszy program w Pythonie!

Zainstalowałeś Pythona, wpisałeś krótki fragment prawdziwego kodu w IDLE i uruchomiłeś swój pierwszy program w Pythonie. Nie jest to szczególnie złożony kod, ale od czegoś trzeba zacząć. Dobra wiadomość jest taka, że jesteśmy już gotowi stworzyć poważną aplikację biznesową!



Czy wyszło Ci coś innego niż „Zaczynamy!”?

Proces pisania i testowania kodu jest podatny na błędy. Jeśli nie uzyskałeś właściwego wyniku za pierwszym podejściem, to powinieneś zacząć się do tego przyzwyczajać. Wszyscy programiści zajmują się ciągłym poprawianiem błędów w kodzie. Oto kilka rozwiązań, które możesz wypróbować:

- *Jeśli Python zwrócił błąd `invalid syntax`, przejrzyj kod pod kątem niepoprawnej interpunkcji, czyli na przykład brakujących nawiasów. Błędy można stosunkowo szybko wykryć, przeglądając podświetlone elementy w IDLE.*
- *Jeśli Python zwrócił błąd `Python NameError: name ,prin' is not defined`, sprawdź kod pod kątem literówek, czyli błędnego zapisu nazw — w tym wypadku chodzi o polecenie `print`.*
- *Jeśli Python zwrócił błąd `EOL while scanning`, zazwyczaj oznacza to, że brakuje któregoś cudzysłowu wokół ciągu znaków. Upewnij się, czy `,Zaczynamy!'` ujęte jest z obu stron w cudzysłów.*

Nie istnieją głupie pytania

P: Dlaczego uruchamiamy kod poleceniem Run Module?

O: W Pythonie plik z kodem nosi nazwę modułu. Polecenie to oznacza zatem „wykonaj cały kod Pythona z mojego pliku”. Moduły służą ponadto do organizowania kodu, o czym jeszcze sobie powiemy.

P: Czym są konkretnie wejścia i wyjścia?

O: W tej chwili używamy prostych wejść i wyjść. Naszym wyjściem (czy też wynikiem) jest tekst wygenerowany przez program i wyświetlany w oknie powłoki Pythona. Wejścia są z kolei tekstem, który program uzyskuje od użytkownika za pośrednictwem okna powłoki. Mówiąc ogólniej, istnieją najróżniejsze rodzaje wejść i wyjść, takie jak dane wejściowe z myszy lub ekranu dotykowego czy dane wyjściowe graficzne i dźwiękowe.

P: Rozumiem, że polecenie `print` (drukuj) służy do wyświetlania tekstu, ale skąd ta nazwa? Na pierwszy rzut oka wygląda to, jakby coś miało zostać wydrukowane.

O: Dawno, dawno temu bardziej prawdopodobne było, że komputer będzie zwracał wyniki w formie drukowanej, a nie na ekranie, więc sama ta nazwa miała więcej sensu. Rzecz jasna Python jest na tyle młodym językiem, że trudno uzasadnić użycie tej nazwy, ale już w wielu językach przyjęło się, że termin `print` służy do wyświetlania wyniku na ekranie — w tym wypadku w oknie powłoki. Nawiasem mówiąc, zetknąłeś się już również z poleceniem `input`, które pobiera dane wejściowe od użytkownika za pośrednictwem powłoki Pythona.

P: Czy polecenie `print` jest jedynym sposobem na uzyskanie wyniku w Pythonie?

O: Nie, to tylko najprostszy sposób. Komputery i języki programowania znakomicie radzą sobie z przekazywaniem danych wejściowych i wyjściowych, a Python nie jest wyjątkiem. Zarówno w Pythonie, jak i w większości innych języków wyniki można przekazywać do stron internetowych, sieci, plików na urządzeniach magazynujących, urządzeniach graficznych, dźwiękowych i innych.

P: Dobrze, czyli kiedy używam tego polecenia, piszę coś w rodzaju `print('no cześć')`. Co tu się właściwie dzieje?

O: Wykorzystujesz tutaj pewne funkcje, używane przez Pythona do drukowania. Mówiąc konkretniej, nakazujesz funkcji o nazwie `print`, aby pobrała tekst zawarty w cudzysłowie i zwróciła go powłoce Pythona. Nieco później omówimy sobie bliżej, czym właściwie są funkcje, czym jest tekst i tak dalej, ale na razie zapamiętaj, że możesz skorzystać z tej funkcji, kiedy chcesz coś wyświetlić w powłoce.

Python bez tajemnic

Dzisiejszy wywiad: czy traktować cię poważnie?

Rusz głową: Witaj, Python! Nie możemy się doczekać, aby cię bliżej poznać i dowiedzieć się, o co właściwie chodzi z tobą.

Python: Mnie też miło.

Rusz głową: Swoją nazwę dostałeś po grupie komediowej i znany jesteś jako język dla żółtodziobów. Dlaczego właściwie ktokolwiek miałby cię brać na poważnie?

Python: No spójrzmy. Jestem używany do różnych zadań, od zarządzania liniami produkcyjnymi układów scalonych, przez tworzenie aplikacji pomagających w powstawaniu wielkich produkcji filmowych (może słyszeliście o niejakim George'u Lucasie), po obsługiwanie interfejsów systemów kontroli ruchu lotniczego. Mógłbym tak bez końca… Czy brzmi to wystarczająco poważnie?

Rusz głową: Skoro jesteś takim poważnym językiem, to jak to możliwe, że początkujący mogą cię z łatwością używać? To wszystko, o czym wspomniałeś, wydaje się dość skomplikowane. Chcesz nam powiedzieć, że nie potrzeba trudnego, złożonego języka, żeby robić takie rzeczy?

Python: Zarówno początkujący, jak i *profesjonaliści* doceniają mnie między innymi dlatego, że mój kod jest dość prosty i czytelny. Widzieliście kiedyś język taki jak Java? Ble. Ile trzeba się namęczyć, żeby wyświetlić proste „Witaj, świecie!". A w Pythonie to jedna linijka kodu.

Rusz głową: No dobrze, jesteś czytelny, to świetnie. Ale co to właściwie oznacza?

Python: Skoro już wspomniałem o Javie, posłużę się przykładem. Wyobraźmy sobie, że chcemy powiedzieć użytkownikowi „Cześć!". W Javie robi się to tak:

```
class PowiedzCzesc {
   public static void main(String[] args) {
      System.out.println(„Cześć!");
   }
}
```

To całkiem sporo i według mnie ten kod jest zupełnie nieczytelny, zwłaszcza dla kogoś, kto się dopiero uczy. Co to wszystko oznacza? Czy to wszystko jest naprawdę potrzebne? Spójrzmy teraz na moją wersję, oczywiście napisaną w Pythonie:

```
print('Cześć!')
```

Chyba zgodzisz się, że tak jest prościej i czytelniej. Każdy może rzucić okiem na tę linijkę kodu i domyślić się, co ona robi. To jednak bardzo prosty przykład. Python ogólnie jawi się jako przejrzysty, niemal naturalny i spójny…

Rusz głową: Spójny? A co to znaczy?

Python: Spójność można rozumieć tak, że w języku nie czai się wiele niespodzianek. Innymi słowy, jeśli rozumiesz już odrobinę język, to inne rzeczy działają w nim tak, jak się spodziewasz. Nie wszystkie języki takie są.

Rusz głową: Wróćmy jeszcze do wcześniejszego wątku. Wspomniałeś o dość niszowych sprawach, takich jak kontrola lotów, produkcja układów scalonych czy sterowanie statkami kosmicznymi. Wszystko to brzmi tak, jakbyś miał zastosowanie głównie w przemyśle i specjalistycznych branżach. Czy Python to aby na pewno dobry wybór dla naszych czytelników?

Python: O statkach kosmicznych akurat nic nie mówiłem. Pozostałe przykłady podałem jako coś, co *możesz uznać za poważne*, skoro masz wątpliwości co do tego, czy Python to poważny język. Tymczasem z Pythona standardowo korzysta się w tworzeniu witryn internetowych, gier, a nawet innych programów komputerowych.

Rusz głową: Jest jeszcze jedna kwestia. Właśnie poinformowano mnie, że istnieją dwie wersje Pythona, które na dodatek… Jak by to uprzejmie ująć? Dwie wersje, które *nie są ze sobą kompatybilne*. Co w tym spójnego?

Python: Języki mają to do siebie, że rozrastają się i ewoluują. Owszem Python występuje w wersji 2 i 3. W wersji 3 znajdują się nowe elementy, których nie było w wersji 2, ale istnieją sposoby na zapewnienie wstecznej kompatybilności. Już mówię, co i jak…

Rusz głową: …i tu kończy się nasz czas. Już wkrótce jednak ponownie cię przydybiemy… To znaczy, znajdziemy okazję, żeby kontynuować tę rozmowę.

Python: Dzięki, nie mogę się doczekać. Chyba.

Wypróbujesz mój nowy Dyrdymałomat, a jak nic zaczniesz gadać jak szef i te chłopy z marketingu...

Czas podejść do sprawy poważnie i napisać praktyczną aplikację w Pythonie. Spójrz na kod Dyrdymałomatu — powinien zrobić na Tobie wrażenie.

Spokojnie

Serio, wyluzuj! Twoim celem jest teraz wchłonięcie tej wiedzy przez osmozę. Przejrzyj wszystkie linijki kodu, przeczytaj opisy ich działania i przyswój to sobie. W kodzie znajdziesz komentarze oznaczone krzyżykami #, które pomogą Ci lepiej zrozumieć, co robi kod. Potraktuj je jak pomocny pseudokod. Do kwestii komentarzy wrócimy nieco dalej. Kiedy już poczujesz, że orientujesz się, co robi ten kod, przejdź na kolejną stronę, aby zapoznać się ze szczegółami.

1 **# powiedz pythonowi, że będziemy używać funkcji losowania**
 # w tym celu zaimportujemy moduł random

```
import random
```

2 **# utwórz trzy listy, jedną z czasownikami,**
drugą z przymiotnikami i imiesłowami,
 # a trzecią z rzeczownikami

```
czasowniki = ['Pozyskać', 'Zsynchronizować', 'Zresetować',
              'Zgamifikować', 'Zdecyfryzować', 'Zaadaptować']
przymiotniki_imieslowy = ['przetestowane', 'freemiumowe',
              'hiperlokalne', 'zsiloizowane', 'blockchainowe',
             'zorientowane na użytkownika', 'osadzone w chmurze',
              'oparte na API']
rzeczowniki = ['aktywa', 'zasoby',
               'procedury', 'generatory leadów', 'algorytmy',
               'procesy', 'punkty krytyczne', 'paradygmaty']
```

3 **# wybierz po jednym czasowniku, przymiotniku (lub imiesłowie)**
i rzeczowniku z każdej listy

```
czasownik = random.choice(czasowniki)
przymiotniki_imieslowy = random.choice(przymiotniki_imieslowy)
rzeczownik = random.choice(rzeczowniki)
```

4 **# teraz utwórz zdanie poprzez „dodanie" tych słów do siebie**

```
zdanie = czasownik + ' ' + przymiotnik_imieslow + ' ' +
rzeczownik
```

5 **# zwróć zdanie**

```
print(zdanie)
```

Dyrdymałomat

Mówiąc w skrócie, ten program wykorzystuje trzy listy wyrazów, losowo wybiera po jednym wyrazie z każdej z nich i łączy je w zdanie (które mogłoby paść na zebraniu w korporacji), a następnie je wyświetla. Nie martw się, jeśli nie rozumiesz żadnego aspektu działania tego programu. Bądź co bądź jesteś dopiero na 25. stronie 600-stronicowej książki. Chodzi tutaj jedynie o to, żeby zaznajomić się z kodem:

1. Instrukcja `import` mówi Pythonowi, że będziemy korzystać z dodatkowej, wbudowanej funkcjonalności, wchodzącej w skład pythonowego modułu `random`. Potraktuj to jako rozszerzenie możliwości kodu, w tym wypadku poprzez dodanie możliwości losowego wybierania elementów. Działanie tego polecenia omówimy bardziej szczegółowo w dalszej części książki.

2. Musimy następnie utworzyć trzy listy. Zadeklarowanie listy jest proste — wystarczy umieścić każdy element listy w cudzysłowie, a całość objąć nawiasami kwadratowymi:

```
czasowniki = ['Pozyskać', 'Zsynchronizować', 'Zresetować',
              'Zgamifikować', 'Zdecyfryzować', 'Zaadaptować']
```

   Zauważ, że przypisujemy każdą listę określonej nazwie, takiej jak `czasowniki`, aby móc się do niej później odnieść w kodzie.

3. Musimy następnie losowo wybrać po jednym wyrazie z każdej listy. W tym celu użyjemy metody `random.choice`, która losowo wybiera jeden element z listy. Następnie przypisujemy ten element do odpowiedniej nazwy (`rzeczownik`, `przymiotnik_imieslow`, `czasownik`), aby móc się do niego później odnieść.

4. Musimy następnie utworzyć zdanie, co robimy poprzez „sklejenie" trzech elementów (czasownika, przymiotnika lub imiesłowu i rzeczownika) — w Pythonie można tak zrobić przy użyciu znaku dodawania. Zauważ też, że musieliśmy wstawić spacje między wyrazami. Bez tego uzyskalibyśmy zdania w rodzaju „Zgamifikowaćfreemiumoweprocesy".

5. Na koniec zwracamy wynik powłoce Pythona instrukcją `print` i gotowe! Możesz się już swobodnie posługiwać biurową nowomową.

Hasła do wykorzystania na kolejnym zebraniu

Zaadaptować blockchainowe algorytmy

Zgamifikować zorientowane na użytkownika zasoby

Pozyskać freemiumowe aktywa

Zresetować hiperlokalne paradygmaty

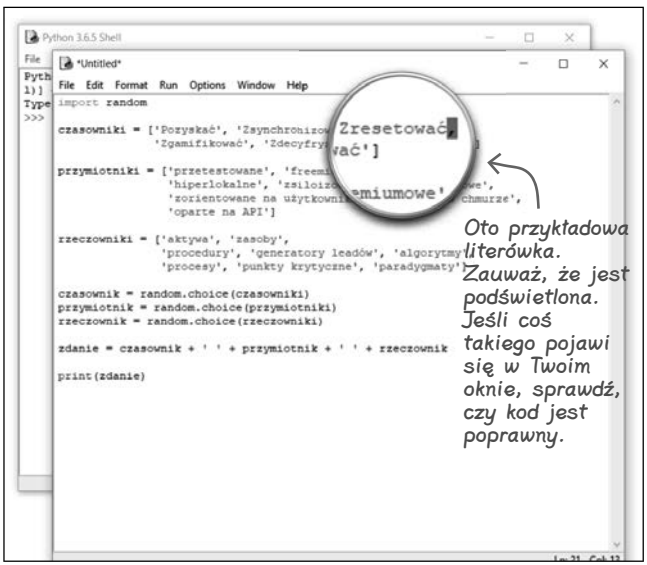
Zsynchronizować zsiloizowane procesy

random.choice jest inną wbudowaną funkcją Pythona. Więcej na temat takich funkcji przeczytasz w dalszej części książki.

Informatycy nazywają takie sklejanie tekstu „konkatenacją". To dość przydatne słowo, z którym zetkniesz się w tej książce jeszcze wielokrotnie.

Wprowadzenie kodu do maszyny

Wprowadziłeś już jeden program do IDLE, ale powtórzmy ten proces. Utworzymy teraz nowy plik Pythona poleceniem *File/New File*. Zrób to i przepisz kod sprzed dwóch stron do edytora.



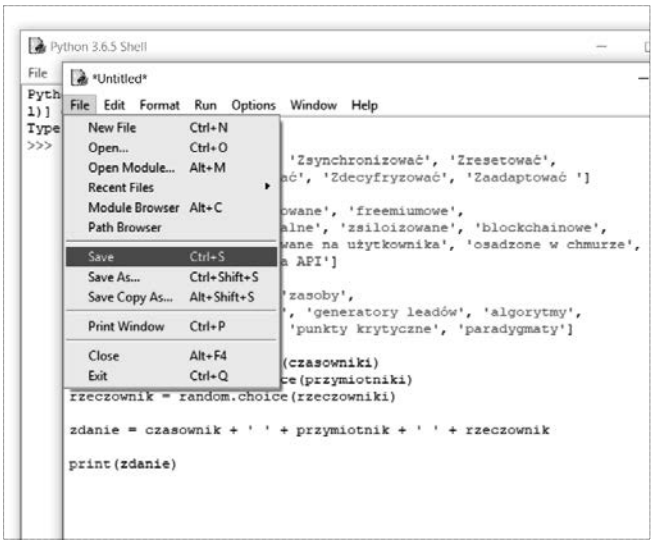
Oto przykładowa literówka. Zauważ, że jest podświetlona. Jeśli coś takiego pojawi się w Twoim oknie, sprawdź, czy kod jest poprawny.

Zwróć szczególną uwagę na wyrazy i interpunkcję. Obydwoma tymi aspektami kodu zajmiemy się szczegółowo w kolejnych rozdziałach, ale na razie wystarczy, że się z nimi ogólnie zaznajomisz.

Zauważ, że IDLE podświetla często występujące błędy. Błędy — w zależności od ich charakteru — mogą się pojawić podczas wpisywania kodu lub przy jego uruchamianiu. Tutaj podświetlony jest błąd polegający na braku cudzysłowu. Jeśli widzisz jakiekolwiek błędy, powinieneś przejrzeć swój kod i je poprawić.

Celowo wprowadziłem tutaj błąd, aby pokazać, co robi IDLE. Nie powinien się on jednak pojawić u Ciebie, jeśli dokładnie przepiszesz kod, który znajduje się na wcześniejszej stronie.

Po wpisaniu całego kodu zapisz plik. Kliknij *File/Save* i podaj nazwę pliku *phraseomatic.py*.



Pamiętaj, że IDLE oznacza kolorami fragmenty kodu o różnych funkcjach.

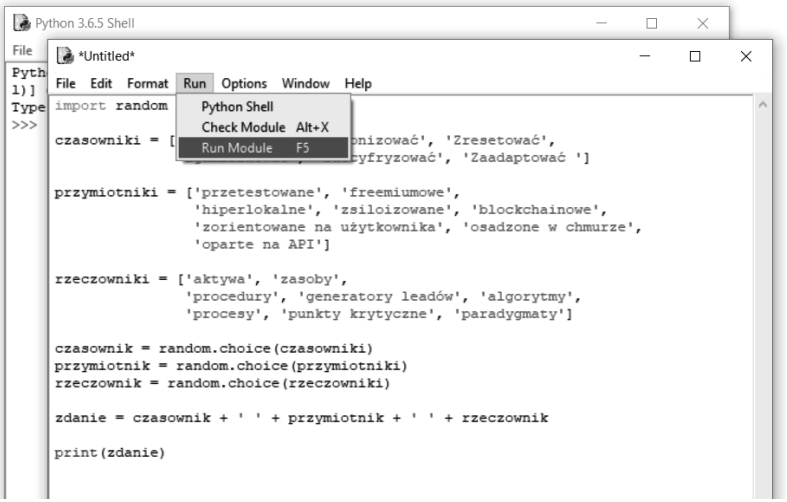
IDLE wprowadza także odpowiednie wcięcia po naciśnięciu klawisza Enter.

Zauważ, że można użyć dodatkowych białych znaków (czyli spacji i znaków nowej linii) do zwiększenia czytelności kodu. Python w dużej mierze ignoruje takie znaki (z pewnym wyjątkiem, który omówimy dalej).

Jazda próbna

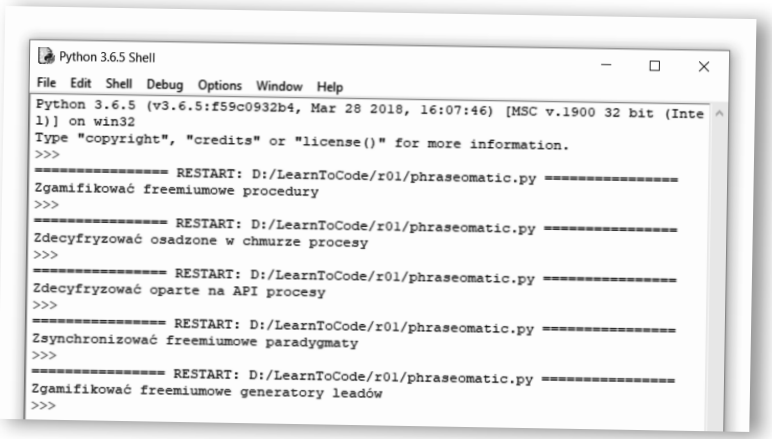
Czas odpalić Dyrdymałomat. Przejdźmy przez cały ten proces ponownie, aby na dobre zapadł Ci w pamięć. Po zapisaniu kodu kliknij *Run/Run Module*. Spójrz następnie na wynik w oknie powłoki i zobacz, co powiedzieć na zebraniu.

Pamiętaj, że kod uruchamia się poleceniem Run Module z menu Run. Jeśli nie zapisałeś swojego kodu, IDLE sam Cię o to poprosi.



Zgamifikujemy przetestowane paradygmaty!

Kilkakrotnie odpaliliśmy Dyrdymałomat; oto nasze wyniki.



Aby ponownie uruchomić Dyrdymałomat, kliknij okno z kodem, a następnie ponownie kliknij polecenie Run/Run Module.

CELNE SPOSTRZEŻENIA

- Aby napisać kod, musisz najpierw rozłożyć problem na prosty zbiór zadań, które ten problem rozwiązują.
- Ten zbiór zadań nazywamy algorytmem lub — mniej formalnie — przepisem na rozwiązanie.
- Działania są przedstawione w formie instrukcji wykonywania prostych zadań, podejmowania decyzji lub kontrolowania przepływu algorytmu poprzez powtarzanie części kodu.
- Myślenie komputacyjne jest podejściem do rozwiązywania problemów, które wyrosło na gruncie informatyki.
- Kodowanie polega na przekształceniu kroków algorytmu na język programowania, który może zostać wykorzystany przez komputer.
- Algorytmy czasami są wyrażane w bardziej zrozumiałym dla ludzi pseudokodzie, zanim zostaną przełożone na rzeczywisty język programowania.
- Języki programowania stworzone są z myślą o opisywaniu zadań komputerom.
- Języki naturalne są słabymi językami programistycznymi ze względu na liczne wieloznaczności.
- Istnieje wiele języków programowania. Każdy z nich ma swoje zalety i wady, ale języki te są jednakowe pod względem obliczeniowym.
- Nazwa Pythona nie wzięła się od węża, lecz od zamiłowania twórcy tego języka do grupy Monty Python.
- Zarówno nowi, jak i doświadczeni programiści doceniają przejrzystość i spójność Pythona.
- Istnieją dwie wersje Pythona — 2 i 3. W tej książce koncentrujemy się na wersji 3, ale w większości przypadków rozbieżności między nimi są minimalne.
- Kod Pythona jest wykonywany przez interpreter, który przekłada wysokopoziomowy kod Pythona na niskopoziomowy kod maszynowy. Kod ten może zostać wykonany bezpośrednio przez Twój komputer.
- Python daje Ci do dyspozycji edytor IDLE, który jest dostosowany do kodu pisanego w tym języku.
- Możesz zwiększyć czytelność kodu programu pythonowego, używając białych znaków.
- `input` i `print` to dwie funkcje, które pozwalają na posługiwanie się danymi wejściowymi i wejściowymi w powłoce Pythona.

Zaostrz ołówek
Rozwiązanie

Prawdziwe przepisy nie wskazują jedynie tego, co zrobić, ale także wymieniają przedmioty potrzebne do przyrządzenia określonego dania, w tym między innymi przybory i urządzenia kuchenne, oraz składniki. Jakie przedmioty wykorzystaliśmy w swoim przepisie na łowienie ryb?

1. Załóż robaka na haczyk.
2. Zarzuć przynętę do wody.
3. Przyglądaj się spławikowi, dopóki się nie zanurzy.
4. Wyciągnij rybę.
5. Jeśli skończyłeś wędkować, idź do domu; w przeciwnym razie wróć do punktu 1

Zaostrz ołówek
Rozwiązanie

W pierwszej kolejności musisz zrozumieć, że komputery robią dokładnie to, co im mówisz — ani mniej, ani więcej. Przyjrzyj się naszemu przepisowi na łowienie ryb z poprzedniej strony. Gdybyś był robotem i postąpił dokładnie według tych instrukcji, jakie problemy mógłbyś napotkać? Czy sądzisz, że ten przepis naprawdę pozwoliłby Ci złowić rybę?

- ☑ A. Jeżeli w wodzie nie ma ryb, to będziesz zajmował się wędkowaniem przez bardzo długi czas (mniej więcej całą wieczność).
- ☑ B. Nie weźmiesz pod uwagę tego, że robak mógłby zlecieć z haczyka i że będziesz musiał założyć nowego.
- ☑ C. Co się stanie, jeśli robaki się skończą?

Wygląda na to, że „wszystkie powyższe" jest poprawną odpowiedzią.

- ☑ D. Czy wskazaliśmy, co zrobić z rybą po wyciągnięciu jej z wody?
- ☑ E. W tym przepisie nie ma ani jednego słowa o wędce.
- ☑ F. *Czy wiadomo konkretnie, jak wygląda dobre zarzucenie? Czy trzeba ponownie zarzucić wędkę, jeśli haczyk wyląduje na lilii wodnej?*

  Kiedy spławik zanurza się pod wodą, na ogół trzeba wykonać zacięcie, zanim zacznie się wyciągać rybę. W przepisie nie ma o tym mowy.

  Skąd mamy wiedzieć, że skończyliśmy wędkować? Po godzinie na zegarku? Po tym, że skończyły się robaki? Po czymś innym?

  W przepisie pojawia się wiele założeń. Zapewne przyszło Ci do głowy wiele innych instrukcji, których nie podano w przepisie.

Magnesiki kodu. Rozwiązanie

Poćwiczmy trochę układanie przepisów algorytmów. Na drzwiach lodówki ułożyliśmy z magnesów algorytm od babci, pozwalający na przyrządzenie omletu z trzech jaj, ale ktoś postanowił zabawić się naszym kosztem i wszystko pomieszał. Czy możesz ułożyć magnesiki we właściwej kolejności, tak aby algorytm zadziałał? Miej na uwadze, że babcia robi dwa rodzaje omletów: zwykły i z serem. **Koniecznie sprawdź poprawną odpowiedź na końcu rozdziału.**

Oto magnesiki we właściwej kolejności!

Istnieje kilka innych poprawnych wariantów. Ważne jest, żebyś zrozumiał moje rozwiązanie, a jeśli Twoje jest odmienne, to powinno mieć logiczny sens.

```
Podgrzej patelnię
```

```
Rozbij trzy jajka do miski
```

Zaczynamy od przygotowania wszystkiego: podgrzania patelni i rozbicia trzech jajek.

```
Dopóki żółtko nie jest w pełni zmieszane z białkiem:
```

Ubijamy jajka, aż białko całkowicie zmiesza się z żółtkiem.

```
    Ubij jajka
```

Zauważ, że ubijanie jajek ułożone jest po wcięciu, aby wskazać, że należy tę czynność wykonywać, dopóki jajka nie są całkowicie ubite. Jeśli wskazałeś to w inny sposób, to nie ma problemu.

```
Przełóż jajka na patelnię
```

Przelewamy następnie jajka na podgrzaną patelnię.

```
Dopóki jajka nie są w pełni ścięte:
```

Smażymy, aż się w pełni zetną.

```
    Mieszaj jajka
```

Wstawiliśmy mieszanie jajek za wcięciem, aby wskazać, że należy je mieszać, dopóki nie są w pełni ścięte. Jeśli wskazałeś to w inny sposób, to nie ma problemu.

```
Jeśli chcesz omlet z serem:
```

Jeśli chcesz omlet z serem, dodaj go.

```
    Posyp serem
```

Dodawanie sera również znajduje się za wcięciem, ponieważ robimy to tylko wtedy, kiedy chcemy zrobić omlet z serem.

```
Zdejmij patelnię z ognia
```

Tak czy inaczej, zdejmujemy patelnię z ognia i przekładamy jajka na talerz.

```
Przelej jajka na patelnię
```

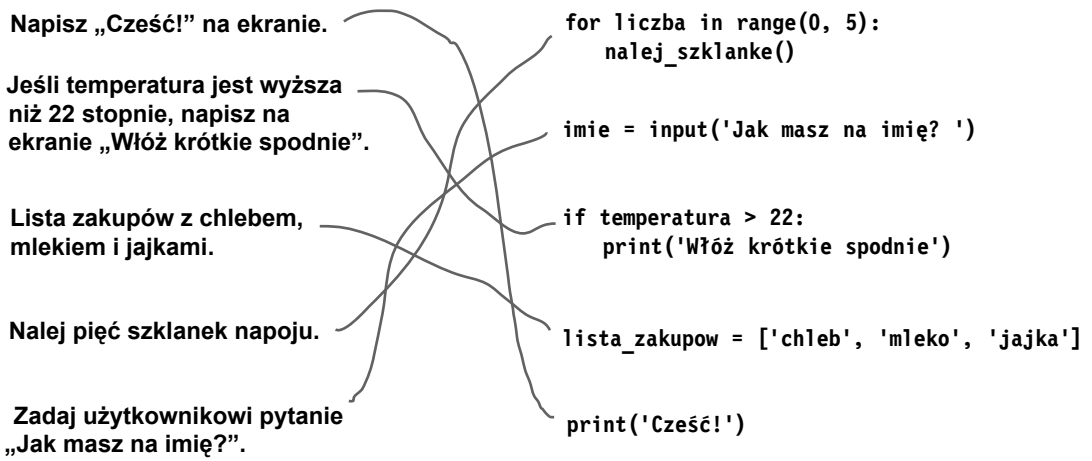
```
Podaj
```

Wreszcie podajemy do stołu.

POMIDOR, POMIDOR

ROZWIĄZANIE

Po lewej widnieją instrukcje napisane po polsku, a po prawej ich odpowiedniki w języku programowania. Narysuj linie łączące obydwie wersje instrukcji. Pierwszą sam narysowałem.



Skorowidz

F

G

H

I

P

R

S

Ś

T

U

W

Z

Ż

O'REILLY®

Rusz głową!
Nauka programowania

Programista to szczególny typ specjalisty. Jeśli uważasz, że myśli w inny sposób niż tak zwani normalni ludzie, to masz rację. Dobra wiadomość jest taka, że i Ty możesz się nauczyć myślenia komputacyjnego — umiejętności, która przydaje się niezależnie od charakteru rozwiązywanego problemu, środowiska czy języka programowania. Tylko w ten sposób można od początku nauki programowania pisać przejrzysty, uporządkowany, znakomity kod, zgodny z najlepszymi praktykami wypracowanymi przez mistrzów. Innymi słowy: pracować jak profesjonalny programista.

Ta książka jest niezwykłym podręcznikiem programowania. Być może wygląda nieco dziwacznie, ale prędko się przekonasz, że to podręcznik wyjątkowo skuteczny: w końcu jego formuła została opracowana na podstawie najlepszych osiągnięć neurologii i kognitywistyki. Dzięki temu Twój mózg się zaangażuje i błyskawicznie przyswoi zasady programowania w Pythonie. Autor wykorzystał oczywistą prawdę, że najszybciej uczymy się wtedy, gdy uwzględnimy specyfikę działania własnego mózgu! Najpierw więc się zainteresujesz, potem zaangażujesz, wreszcie przygotujesz sobie warsztat pracy, czyli zainstalujesz Pythona. Później zaczniesz ćwiczyć myślenie komputacyjne i oczywiście napiszesz swój pierwszy program. A dalej będzie coraz ciekawiej...

W tej książce znajdziesz między innymi:

- istotne koncepcje programistyczne
- zasady programowania w Pythonie
- funkcje i rekurencję
- programowanie obiektowe
- tworzenie API dla aplikacji internetowych
- widgety i zdarzenia

Neurony płoną.
Emocje szaleją.
Tak napiszesz
kod godny mistrza!

Eric Freeman — jest informatykiem o imponującym doświadczeniu zawodowym: pracował między innymi dla The Walt Disney Company, O'Reilly Media, NASA i w kilku startupach. Jeśli tylko używasz komputera osobistego, na pewno korzystasz z dzieł jego intelektu. Freeman od 15 lat pisze o wielu dziedzinach informatyki: zarówno o podstawach tworzenia witryn internetowych, jak i o wysokopoziomowym projektowaniu oprogramowania. Obecnie jest prezesem spółki WickedlySmart. Wraz ze swoimi ukochanymi kobietami: żoną i córką mieszka w Austin.

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

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-4697-0

9 788328 346970

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 99,00 zł