

# Rusz głową!

# Nauka

# programowania

Uważaj na  
powszechnie  
występujące  
błędy i pułapki



Unikaj  
zawstydzających  
błędów składniowych

Gimnastykuj  
swój umysł,  
rozwiązując  
około 130  
łamigłówek  
i ćwiczeń



Przewodnik  
po kodowaniu  
i myśleniu komputacyjnym

Rozpocznij  
karierę  
programisty



Dowiedz się, dlaczego  
wszystko to, co Twój  
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

© 2019 Helion S.A.

Authorized Polish translation of the English edition of Head First Learn to Code ISBN 9781491958865

© 2018 Eric Freeman

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

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

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

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

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

Helion SA

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: [helion@helion.pl](mailto: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.

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

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

## Spis treści (streszczenie)

	Wstęp	xxv
1	Zaczynamy: <i>myślenie komputacyjne</i>	1
2	Znaj swoje wartości: <i>proste wartości, zmienne i typy</i>	33
3	Decyzyjny kod: <i>wartości logiczne, decyzje i pętle</i>	71
4	Odrobina struktury: <i>listy i iteracje</i>	121
5	Funkcyjny kod: <i>funkcje i abstrakcje</i>	173
4 (część II)	Porządkowanie danych: <i>sortowanie i zagnieżdżona iteracja</i>	217
6	Składanie wszystkiego w całość: <i>tekst, łańcuchy i heurystyki</i>	237
7	Myślenie modułarne: <i>moduły, metody, klasy i obiekty</i>	281
8	Poza iteracją i indeksami: <i>rekurencja i słowniki</i>	329
9	Trwałość danych: <i>zapisywanie i zwracanie plików</i>	379
10	Żądania i odpowiedzi: <i>używanie webowych API</i>	419
11	Interaktywność: <i>widżety, zdarzenia i zachowania emergentne</i>	449
12	Wycieczka do Obiektowa: <i>programowanie obiektowe</i>	505
	Dodatek	555
	Skorowidz	567

## 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 *przechrztyć* mózg, żeby zaczął myśleć, że Twoje życie istotnie zależy od umiejętności programowania?



Dla kogo jest ta książka?	xxvi
Wiemy, co myślisz	xxvii
Czytelnika książek z serii Rusz głową! traktujemy jak ucznia	xxviii
Metapoznanie: myślenie o myśleniu	xxix
Oto co MY zrobiliśmy	xxx
A co TY możesz zrobić, żeby zapanować nad swoim mózgiem?	xxxi
Przeczytaj to	xxxii
Podziękowania	xxxvii
Recenzenci	xxxviii

# 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	2
Jak działa kodowanie?	6
Ale po jakiemu to?	7
Świat języków programowania	8
Pisanie i uruchamianie kodu w Pythonie	13
Bardzo krótka historia Pythona	15
Python — pierwsza próba	18
Zapisywanie wyników swojej pracy	20
Gratulacje! Właśnie napisałeś swój pierwszy program w Pythonie!	21
Dyrzymałomat	25
Wprowadzenie kodu do maszyny	26

## 2

## Proste wartości, zmienne i typy

## 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**.

Kodowanie kalkulatora wieku psa	34
Od pseudokodu do kodu	36
Krok 1. Pobranie danych wejściowych	37
Jak działa funkcja input	38
Używanie zmiennych do zapisywania i przechowywania wartości	38
Przypisanie danych wejściowych użytkownika do zmiennej	39
Krok 2. Zbieranie kolejnych danych	39
Czas uruchomić kod	40
Wprowadzanie kodu	43
Ze zmiennymi na głęboką wodę	44
Wyraźmy to lepiej	45
Zmienne nazywają się tak nie bez powodu	46
Pierwszeństwo operatorów kluczem do szczęśliwego życia	47
Obliczanie z wykorzystaniem pierwszeństwa	48
Ręce z klawiatury!	51
Krok 3. Obliczanie wieku psa	52
Houston, mamy problem!	53
Ludzką rzeczą jest błędzić	54
Jeszcze trochę debugowania...	56
Co to za typy?	58
Naprawianie kodu	59
Houston, wystartowaliśmy	60
Krok 4. Czytelny wynik	61
Końcowa jazda próbna	62



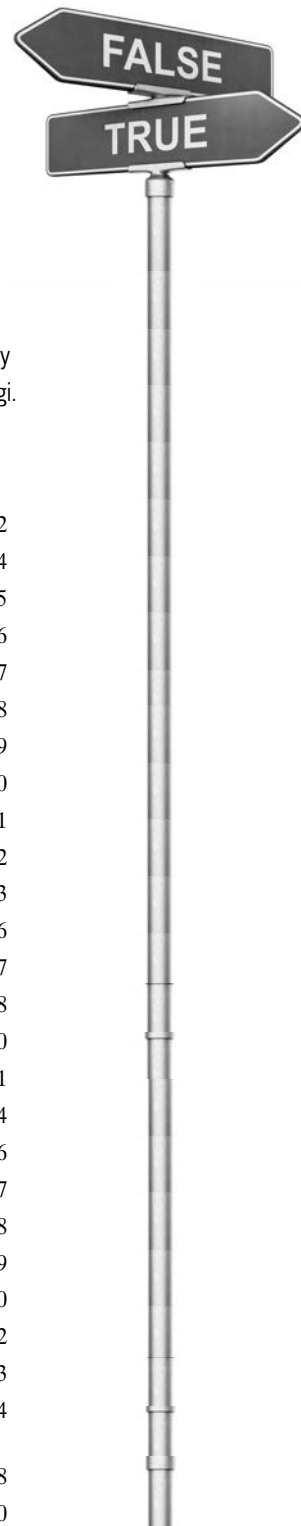
## 3

## Wartości logiczne, decyzje i pętle

**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 być 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!

Może chciałbyś zagrać?	72
Najpierw projekt	74
Wybór komputera	75
Jak używać liczb losowych?	76
Kolejny krok...	77
Prawda czy fałsz?	78
Wartości logiczne	79
Podejmowanie decyzji	80
Decyzje, decyzje...	81
Wracamy do gry	82
Pobranie wyboru użytkownika	83
Sprawdzenie wyboru użytkownika	86
Wprowadzenie kodu wykrywającego remis	87
Kto wygrał?	88
Jak zaimplementować logikę gry?	90
Więcej o operatorach logicznych	91
Wyświetl, kto wygrał	94
A dokumentacja jest?	96
Jak umieszczać komentarze w kodzie?	97
Musimy dokończyć grę!	98
Skąd wiemy, że wybór użytkownika jest niepoprawny?	99
Sprawdzanie i porządkowanie wyrażenia	100
Jak wyświetlać zapytanie cyklicznie?	102
Wykonywanie działań więcej niż raz	103
Jak działa pętla while?	104
Jak użyć while do wyświetlania zapytań aż do otrzymania poprawnego wyboru	108
Gratulacje! Zakodowałeś swoją pierwszą grę!	110



## Listy i iteracje

## 4

**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 dojdiesz do etapu, na którym będziesz chciał stworzyć kod wykorzystujący liczne zbiory danych, obejmujące na przykład wszystkie elementy w koszyku zakupów, nazwy najjaśniejszych 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 obsługiwać się danymi.

Czy pomożesz Bańkom?	122
Jak przedstawić wiele wartości w Pythonie?	123
Jak działają listy?	124
Jak pobrać element listy?	125
Zmiana wartości w liście	125
Jak długa jest ta lista?	127
Pobieranie ostatniego elementu listy	128
Stosowanie indeksów ujemnych Pythona	129
Tymczasem w firmie Bańki S.A...	131
Dyskusja nad biurkiem	133
Jak iterować listę?	134
Usuwanie błędu	135
Krótką jazda próbna	135
Skuteczne usuwanie błędu	136
Szybka jazda próbna	136
Pętla <code>for</code> — preferowana metoda iterowania listy	138
Jak pętla <code>for</code> działa na zakresie liczb?	141
Zastosowania zakresów	142
Generowanie raportu	144
Przetestuj raport o roztworach na bańki	145
Dyskusja nad biurkiem — ciąg dalszy	149
Tworzenie własnej listy od podstaw	152
Dodatkowe działania na listach	153
Przetestuj gotowy kod raportu	157
Najlepsze roztwory to...	157
Testowanie kodu znajdującego najbardziej opłacalny roztwór	161



## 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.

Co jest z tym kodem nie tak?	175
Przekształcenie bloku kodu w FUNKCJĘ	177
Jak użyć utworzonej funkcji?	178
Jak to właściwie działa?	178
Funkcje mogą też ZWRACAĆ wyniki	186
Jak wywołać funkcję z wartością zwrotną?	187
Pierwsze kroki z refaktoryzacją	189
Wykonanie kodu	190
Jak wyabstrahować kod awatara?	191
Pisanie ciała funkcji pobierz_ceche	192
Wywoływanie funkcji pobierz_ceche	193
Musimy porozmawiać o zmiennych...	195
Zasięg zmiennej	196
Kiedy zmienne są przekazywane funkcjom?	197
Wywołanie funkcji wypij_mnie	198
Co ze zmiennymi globalnymi w funkcjach?	201
Zaawansowane parametry: wartości domyślne i argumenty kluczowe	204
Zawsze deklaruj wymagane parametry jako pierwsze!	205
Używanie argumentów kluczowych	206
Jak korzystać z tych opcji?	206





# 4

## Część II

### Sortowanie i zagnieżdżona iteracja

#### Porządkowanie danych

**Czasami domyślna kolejność danych jest po prostu niezadowolająca.** 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 świnie — 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!

Dyskusja nad biurkiem	219
Zrozumieć sortowanie bąbelkowe	220
Pseudokod sortowania bąbelkowego	223
Dyskusja nad biurkiem — ciąg dalszy	224
Implementacja sortowania bąbelkowego w Pythonie	226
Obliczanie numerów rozтворów na bańki	228



## Tekst, łańcuchy i heurystyki

**Składanie wszystkiego w całość**

## 6

**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!

Wprowadzenie do analityki danych	238
Jak obliczyć czytelność?	239
Plan działania	240
Przygotowanie pseudokodu	241
Potrzebujemy tekstu do analizy	242
Przygotowanie funkcji	244
Przede wszystkim: musimy uzyskać łączną liczbę słów w tekście	245
Dyskusja nad biurkiem	246
Obliczanie łącznej liczby zdań	249
Tworzenie funkcji policz_zdania	250
Obliczanie liczby sylab, czyli jak przestałem się martwić i pokochałem heurystykę	256
Dyskusja nad biurkiem — ciąg dalszy	258
Formułowanie heurystyki	259
Pisanie heurystyki	260
Jak liczyć samogłoski?	261
Ignorowanie konsekwentnych samogłosek	261
Pisanie kodu ignorującego konsekwentne samogłoski	262
Pomijanie końcowych liter e i y oraz interpunkcji	264
Krojenie w praktyce	266
Wykończenie kodu heurystyki	268
Implementacja wzoru na czytelność	270
Co dalej?	275

Całkiem ambitna ta książka.




## Moduły, metody, klasy i obiekty

# 7

### Myślenie modułarne

**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.

Dyskusja nad biurkiem	283
Krótkie omówienie modułów	284
Dyskusja nad biurkiem — ciąg dalszy	285
Zmienna globalna <code>__name__</code>	286
Rozmowa nad biurkiem — ciąg dalszy	287
Aktualizacja pliku <code>analize.py</code>	287
Wykorzystanie pliku <code>analize.py</code> jako modułu	289
Dodawanie notek dokumentacyjnych do pliku <code>analize.py</code>	291
Inne moduły Pythona	295
Zaraz... „żółwie”?!	296
Tworzenie własnego żółwia	298
Żółwiowe laboratorium	299
Dodawanie drugiego żółwia	301
Czym w ogóle są te żółwie?	304
Czym są obiekty?	305
Czym jest w takim razie klasa?	306
Klasa określa, co obiekt wie i co może zrobić	307
Jak używać obiektów i klas?	308
Co z metodami i atrybutami?	309
Klasy i obiekty są wszędzie	310
Zorganizujmy wyścigi żółwi	312
Planowanie gry	313
Zacznijmy pisać kod	314
Pisanie kodu przygotowującego	315
Hola, hola!	316
Rozpoczęcie wyścigu	318



Świetna robota! Mogłem bezproblemowo użyć modułu `analize` między innymi dzięki dobrej dokumentacji!



UWAGA! MIEJSCE ZBRODNI

UWAGA! MIEJSCE ZBRODNI

UWAGA! MIEJSCE ZBRODNI

## 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.



Inny sposób wykonywania obliczeń	330
Zróbmy to teraz inaczej...	331
Napiszmy teraz kod na potrzeby obydwu przypadków	332
Ćwiczmy dalej	335
Wykrywanie palindromów rekurencyjnie	336
Pisanie rekurencyjnego wykrywacza palindromów	337
Serwis Aspołecznościowy	348
Wprowadzenie do słowników	350
Co z iterowaniem po słowniku?	352
Wykorzystanie słowników w Serwisie Aspołecznościowym	354
Jak dodać więcej właściwości?	356
Pamiętasz o głównej funkcji Serwisu Aspołecznościowego?	358
Teraz wszystko w Twoich rękach!	360
Czy możemy po prostu zapisywać wyniki?	364
Użyjmy cennego słowa: memoizacja	365
Jak działa funkcja koch?	368
Fraktal Kocha z bliska	370
Ten fraktal nieprzypadkowo nazywa się płatkami	370

## 9

## Zapisywanie i zwracanie plików

## Trwałość danych

**Wiesz już, że można zapisywać wartości w zmiennych, ale po wykonaniu programu — puף! — 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.

Może opowiemy sobie szaloną historię?	380
Jak działają Szalone historie?	382
Krok 1. Odczytanie tekstu historyjki z pliku	385
Jak używać ścieżek plików?	386
Ścieżki bezwzględne	387
Pamiętaj, żeby po sobie posprzątać!	388
Wczytywanie pliku do kodu Pythona	389
Odczyt pliku przy użyciu obiektu plikowego	389
Zrób sobie przerwę	392
Hej, musimy jeszcze skończyć naszą grę!	393
Skąd wiemy, że dotarliśmy do ostatniego wiersza?	395
Sekwencje Pythona mogą nam to ułatwić	395
Wczytywanie szablonu Szalonych historii	396
Przetwarzanie tekstu szablonu	397
Użycie nowej metody łańcucha do naprawienia błędu	399
Naprawienie błędu	400
Problemy z kodem	401
Obsługa wyjątków	403
Bezpośrednie obsługiwanie wyjątków	404
Obsługa wyjątków w Szalonych historiach	406
Ostatni krok: zapisywanie historyjki	407
Aktualizacja pozostałej części kodu	407



## 10

## Używanie webowych API

**Żądania i odpowiedzi**

**Potrąfisz 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.

Sięgnij dalej za pomocą webowych API	420
Jak działają webowe API?	421
Każde webowe API ma adres internetowy	422
Wprowadźmy drobne ulepszenie	425
Aktualizacja	426
Potrzebujemy teraz dobrego webowego API...	427
API z bliska	428
Webowe API przekazują dane przy użyciu JSON-a	429
Przyjrzyjmy się ponownie modułowi <code>requests</code>	431
Składanie wszystkiego w całość: wysłanie żądania do Open Notify	433
Jak używać JSON-a w Pythonie?	434
Używanie modułu <code>JSON</code> z danymi ISS	435
Dodajemy grafikę	436
Tworzymy obiekt ekranu	437
Dodajemy żółwia reprezentującego ISS	439
Żółw może też wyglądać jak stacja kosmiczna	440
Stacja jak stacja, ale co z kodem?	441
Uzupełnienie kodu ISS	442



## 11

## Widżety, zdarzenia i zachowania emergentne

**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. Napijemy symulator sztucznego życia z emergentnymi zachowaniami. Co to właściwie znaczy? Tego dowiesz się na następnej stronie.

Poznaj CUDOWNY ŚWIAT niesamowitego SZTUCZNEGO ŻYCIA	450
Gra w życie z bliska	451
Co chcemy stworzyć?	454
Czy mamy odpowiedni projekt?	455
W jaki sposób utworzymy symulator?	458
Tworzenie modelu danych	459
Obliczanie stanu pokolenia w grze	460
Uzupełnianie kodu modelu	464
Na jakim etapie jesteśmy?	466
Tworzenie widoku	467
Tworzenie swojego pierwszego widżetu	468
Dodawanie pozostałych widżetów	469
Poprawianie układu	470
Rozmieszczanie widżetów w układzie siatki	471
Zajmijmy się kontrolerem	473
Gotów na kolejny nowy styl programowania?	476
Dodajmy handler kliknięć	477
Jak będzie działać przycisk startu/pauzy?	479
Implementacja przycisku startu i pauzy	480
Zdarzenie innego rodzaju	481
Mamy odpowiednią technologię: metodę after	483
Daleko jeszcze?	484
Jak bezpośrednio wprowadzać i edytować komórki?	486
Pisanie handlera widok_siatki	487
Czas dodać predefiniowane wzorce	488
Pisanie kodu handlera OptionMenu	489
Jak zdefiniować wzorce?	491
Pisanie funkcji wczytywania wzorców	492
Jazda próbna	493



## 12

## Programowanie obiektowe

## 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! Przyjrzelśmy 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 nauczaniu 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.

Rozkładanie zadań w inny sposób	506
Jaki jest sens programowania obiektowego?	507
Projektujemy swoją pierwszą klasę	509
Piszemy swoją pierwszą klasę	510
Pisanie metody szczerkanie	513
Jak działają metody?	514
Dodawanie dziedziczenia	516
Implementacja klasy <code>PiesTowarzyszczy</code>	517
Podklasy z bliska	518
<code>PiesTowarzyszczy</code> to też <code>Pies</code>	519
Sprawdzanie obecności relacji IS-A	520
Oddajmy głos psom	523
Nadpisywanie i rozszerzanie metod	524
Witamy w Żargonowie	526
Obiekt może zawierać inny obiekt	528
Projektowanie Hotelu dla Psiaków	531
Implementacja Hotelu dla Psiaków	532
Remont Hotelu dla Psiaków	535
Dodawanie zajęć w hotelu	536
Umiem wszystko, co ty umiesz, czyli polimorfizm	537
Czas nauczyć pozostałe psy chodzenia	538
Moc dziedziczenia	540
Tworzenie usługi wyprowadzania psów	541
Jak mamy zatrudnić wyprowadzacza, skoro nawet nie mamy obiektów osób?	542
Tymczasem na wyścigach żółwi...	545
ROZWIĄZANIE afery z żółwiami	547





Dodatek. Coś na deser

## A

**Dziesięć najważniejszych tematów  
(których nie omówiliśmy)**

**Zdołaliśmy omówić wiele obszarów tematycznych i pomalutku zbliżamy się do końca.** Smutno tak się rozstawać, ale nie wybaczyłbym sobie, gdybym wysłał Cię w szeroki świat bez odrobiny przygotowania. Niestety nie zmieścę 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 wspominałem 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 Listy składane	556
#2 Daty i godziny	557
#3 Wyrażenia regularne	558
#4 Inne typy danych: krotki	559
#5 Inne typy danych: zbiory	560
#6 Kod po stronie serwera	561
#7 Ewaluacja leniwa	562
#8 Dekoratory	563
#9 Funkcje wyższego rzędu i pierwszoklasowe	564
#10 Biblioteki	565

## S

**Skorowidz**

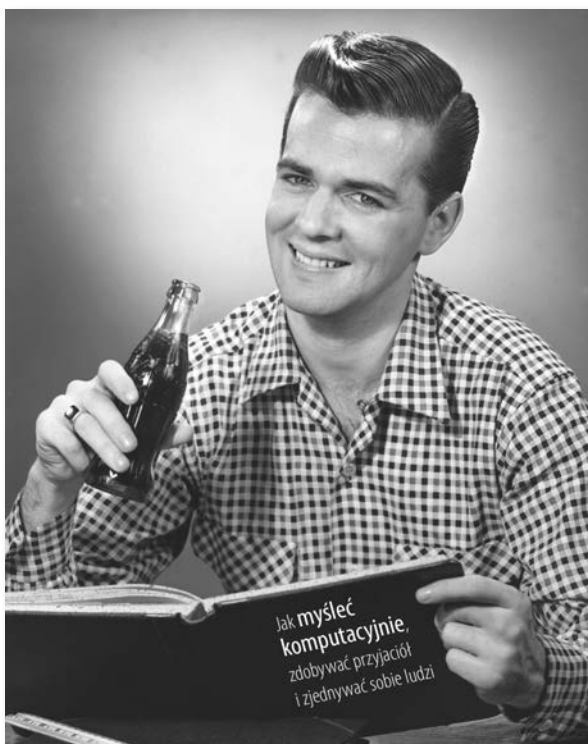
567





# 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: założmy, że chcesz rozłożyć zajęcie, jakim jest łowienie ryb, na prosty zestaw instrukcji dla robota, który zajmie się tym za Ciebie. Oto nasza pierwsza próba:



Rozłożmy proces łowienia ryb na ciąg prostych i zrozumiałych kroków.

1 Załóż robaka na haczyk.

2 Zarzuć przynętę do wody.

3 Przyglądaj się splotkowi, dopóki się nie zanurzy.

4 Wyciągnij rybę.

5 Jeśli skończyłeś wędkować, idź do domu; w przeciwnym razie wróć do punktu 1.

Poniższe kroki wykonujemy po kolei.

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 splotwika 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 szałwiarz 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 wędcie.
- F. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Czy przychodzą Ci do głowy jeszcze jakieś problemy?



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

Odpowiedzi do ćwiczeń z serii „Zaostrz ołówki” znajdują się na końcu rozdziałów.

To ja wykosztowałam się na poważną książkę techniczną, a tu muszę czytać o jakichś przepisach? To nie brzmi zbyt obiecująco. Technicznie też nie.



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”.

**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**.

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.

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).

Już wkrótce wrócimy do tematu pseudokodu...

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



## 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 pomieszało. Czy możesz ułożyć magnesiki we właściwej kolejności, tak aby algorytm zadziałał? Miej na uwadze, że babcia robi dwa rodzaje omletów: zwykły i z serem. **Konieczn**

Ułóż 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

Ubij jajka

Podgrzej patelnię

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 zgłębnimy 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 pożądaną 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.

Rzeczniemy 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('Zarzućam przynętę')  
  
while True:  
    odpowiedz = input('Czy spławik jest zanurzony pod wodą? ')  
    if odpowiedz == 'tak':  
        rusza_sie = True  
        print('Złapała haczyki!')  
        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*.



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

Tych terminów też będziemy używać zamiennie.



## Ale po jakiemu to?

**Język programowania** powinien być traktowany 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:




Pisanie w Objective-C to mój żywioł. Spędzam całe dni na tworzeniu apek na iPhone'a. Podoba mi się to, że Objective-C przypomina C, ale za to jest obiektowy i o wiele bardziej dynamiczny. Uczę się też nowego języka Apple'a — Swift.

Programowanie w Javie wiąże się z myśleniem na poziomie obiektów, a nie kodu niskiego poziomu. Java zajmuje się za mnie sprawami niskopoziomymi, takimi jak zarządzanie pamięcią i wątkami.


Żyję w świecie WordPressa, który bazuje na PHP, więc siłą rzeczy korzystam z tego języka. Niektórzy mówią, że to język skryptowy, ale mogę w nim zrobić wszystko to, czego potrzebuję.




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 zabrzmieć 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.

## Dlaczego wybieramy Pythona

Uwielbiamy Pythona. Jest to bardzo czytelny, przejrzysty język ze świetną obsługą bibliotek, pozwalający na pisanie kodu na potrzeby najróżniejszych dziedzin. Poza tym ma on wspaniałą społeczność użytkowników.

Python znany jest jako jeden z najlepszych języków dla początkujących, ale jego możliwości wzrastają wraz z rozwojem Twoich umiejętności. To jednocześnie „poważny” język – Google, Disney i NASA używają go do tworzenia własnych systemów.



### 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 liniom 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 + '! Wygrałeś/-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! Wygrałeś/-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 liniom 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 + '! Wygrałeś/-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.*

*Losowo wybiera jednego z klientów.*

*Przydziela zmiennej o nazwie smak wartość tekstową 'wanilia'.*

*Wyświetla w konsoli gratulacje z imieniem zwycięzcy. Jeżeli na przykład wygrała Ada, ta linijka kodu wyświetla tekst 'Gratulacje, Ada! Wygrałeś/-eś pucharek lodów!'.*

*Przydziela zmiennej o nazwie pytanie wartość tekstową 'Czy chcesz wisienkę na wierzchu? '.*

*Prosi użytkownika o wpisanie tekstu, który następnie przypisuje zmiennej wisienka. Zauważ, że w konsoli Pythona w pierwszej kolejności pojawia się wartość zmiennej pytanie.*

*Nadaje zmiennej zamowienie wartość tekstową ' pucharek lodów o smaku ', po której podaje wartość zmiennej smak.*

*Jeżeli użytkownik odpowiedział „tak” na pytanie „Czy chcesz wisienkę na wierzchu?”, do zmiennej zamowienie dodany zostaje tekst ' z wisienką na wierzchu'.*

*Wyświetla w konsoli informację o tym, że zamówienie zwycięzcy losowania za moment do niego trafi.*

#### Python Output

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

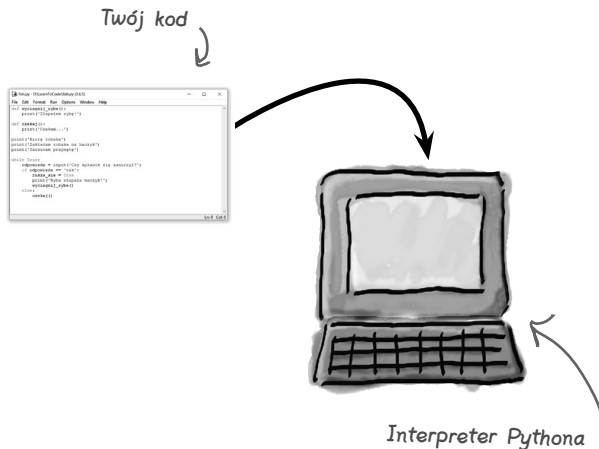
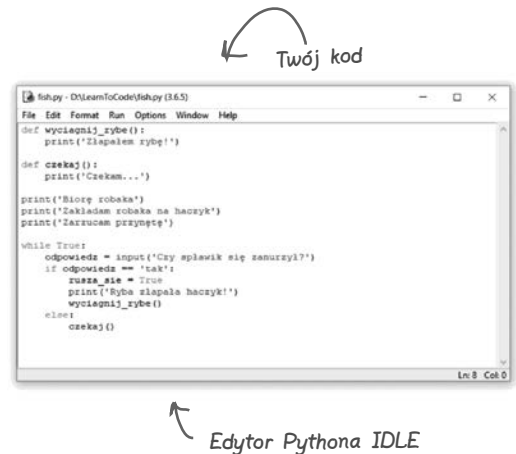
*PS. Jeżeli nie możesz się oprzeć pokusie i chcesz wypróbować ten kod, wpisz `import random` w pierwszej linijce pliku przed jego uruchomieniem. Później dowiesz się, dlaczego tak należy zrobić, ale powinieneś wziąć pod uwagę, że w tej chwili uruchamianie kodu nie jest ani konieczne, ani szczególnie przydatne. Niemniej jednak wiem po prostu, że znajdują 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 przejrzelśmy 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. Rozzeźnijmy 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 istnieje  
grupie pytania

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

**U:** 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?

**U:** Technicznie rzecz ujmując, wszystkie współczesne języki programowania są równorzędne, ponieważ pozwalają na obliczanie tych samych rzeczy, wobec czego teoretycznie rzeczywistość 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?

**U:** 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 myśleniem komputacyjnego? To drugie to chyba coś informatycznego?

**U:** 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 nauczaniu komputera, jak wykonywać takie algorytmy za nas — i tutaj właśnie przechodzimy do kodowania. Kodowanie jest metodą przekazania algorytmu komputerowi (bądź jakimkolwiek 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

- A. Twórca Pythona uwielbiał węże, a już wcześniej stworzył mniej udany język Cobra.
- B. Twórca był ciekawski, więc zadawał wiele pytań. Stąd Python.

- Jak sądzisz, skąd wzięła się nazwa Python?  
Zaznacz poniżej najbardziej prawdopodobną odpowiedź.
- 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 spotkanie Pythona postępuje się wieloma nawązaniami 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 zmyśliłem.*



### 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!



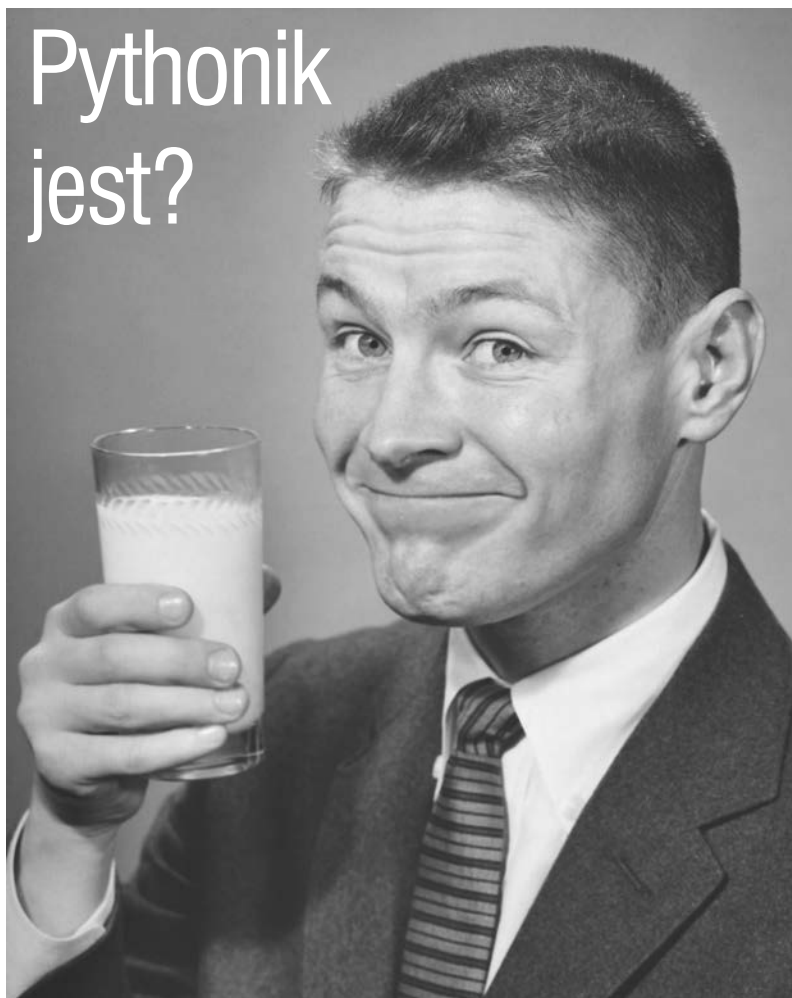
Czyli mamy Pythona 2 i 3. Z której wersji będziemy korzystać i czym się właściwie one różnią?

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, odnieśliśmy 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.

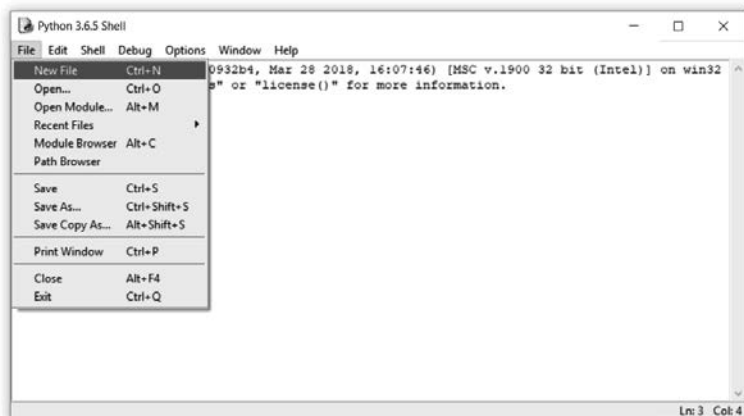
# Python — pierwsza próba

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*.



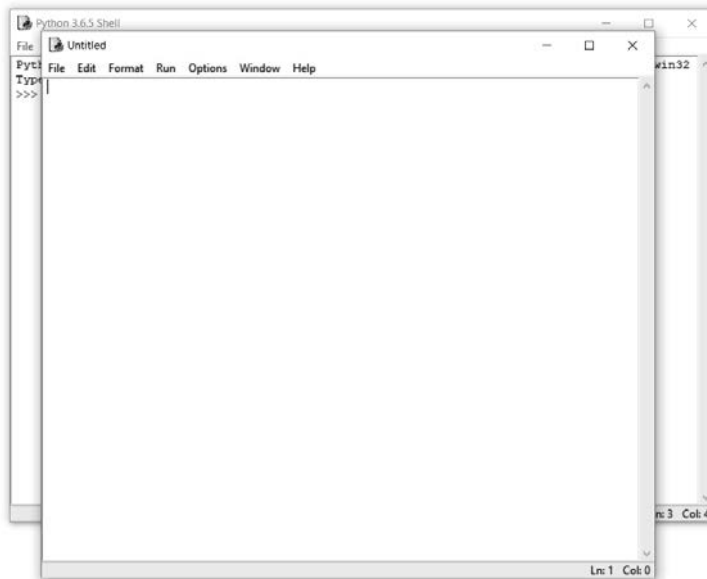
Po uruchomieniu IDLE na ekranie pojawi się interaktywny interpreter, zwany powłoką Pythona. Jeśli jesteś ciekaw, wpisz w nim `1+1` (jeden plus jeden) i naciśnij `Enter`. Więcej na ten temat przeczytasz w następnym rozdziale.

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.

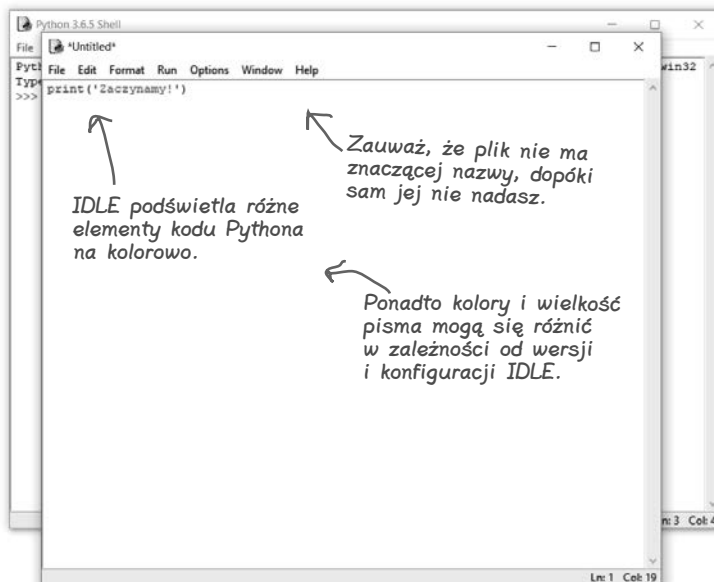
IDLE działa tak jak procesor tekstu, z tym że rozumie kod Pythona i pomaga Ci — podświetla słowa kluczowe Pythona, co ułatwia formatowanie, a niekiedy nawet daje możliwość autouzupełnienia często występujących słów kluczowych.



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

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

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



IDLE podświetla różne elementy kodu Pythona na kolorowo.

Zauważ, że plik nie ma znaczącej nazwy, dopóki sam jej nie nadasz.

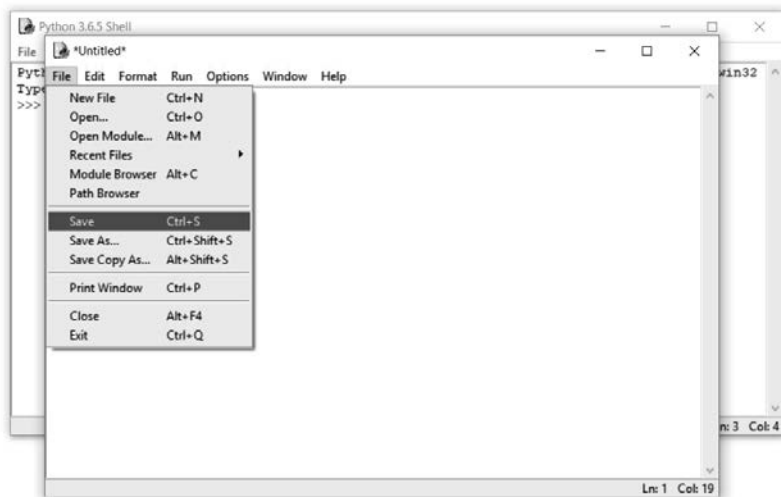
Ponadto kolory i wielkość pisma mogą się różnić w zależności od wersji i konfiguracji IDLE.

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

## Jak zapisywać programy

# 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”.

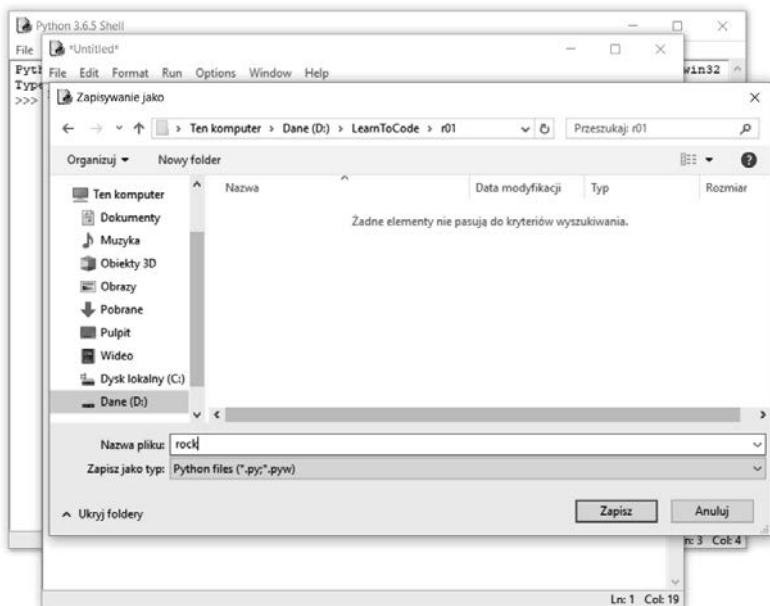


Kod źródłowy, plik źródłowy i program są wymiennymi nazwami plików, w których mieści się kod.

Nadaj swojemu kodowi nazwę i rozszerzenie *.py*. Zdecydujemy 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 zrobisz, rzuć okiem na s. XXXVI we wprowadzeniu.



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

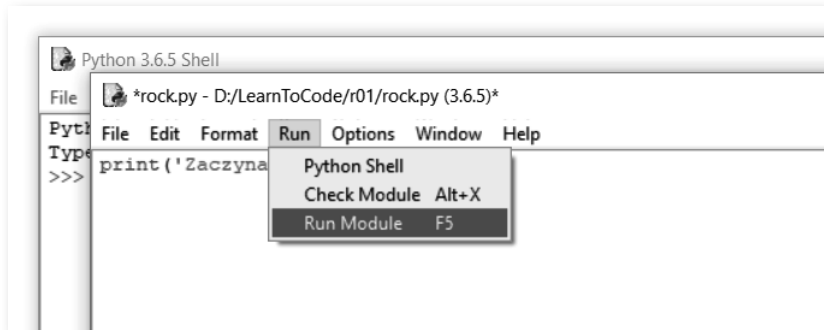




## Jazda próbna

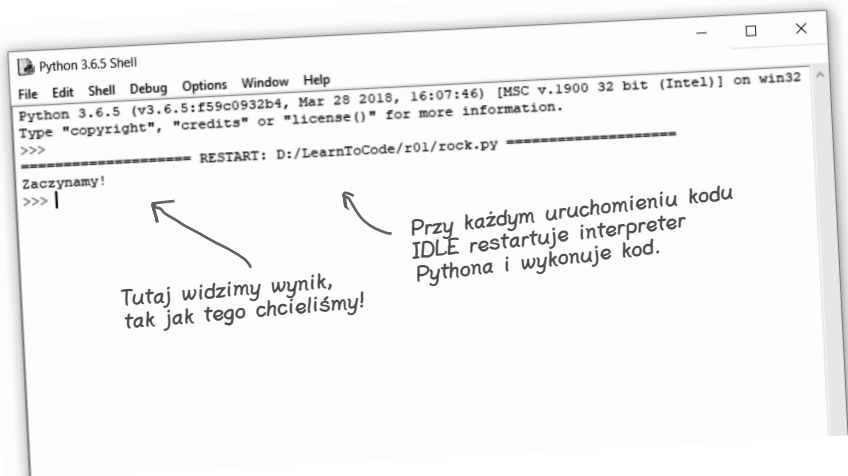
Teraz przechodzimy do rzeczy. Po zapisaniu kodu kliknij polecenie *Run/Run Module* (uruchom/uruchom moduł). Spójrz następnie na okno powłoki Pythona, aby zobaczyć wynik programu.

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 czego 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 'print' 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 cudzysłow.

### 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 wejściowe 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 powłocie 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 powłocie.





## 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 jakimś 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 *profesjoniści* 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?

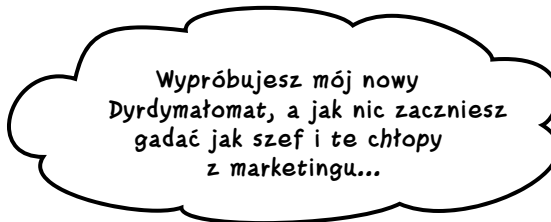
**Python:** Języki mają to do siebie, że rozrastają się i ewoluują. Owszem Python występuje w wersji 2 i 3. W wersji 3 znajdują się nowe elementy, których nie było w wersji 2, ale istnieją sposoby na zapewnienie wstecznej kompatybilności. Już mówię, co i jak...

**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.



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



- 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', 'Zdecyfrzować', 'Zadaptować']  
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)
```



### Spokojnie

Serio, wyluzuj!  
Twoim celem jest  
teraz wchłonięcie  
tej wiedzy przez  
osmozę. Przejrzyj wszystkie  
linijki kodu, przeczytaj opisy  
ich działania i przyswoj 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.

## Dyrzymałomat

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 zaznaczyć 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 powłocie 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 sklejenie tekstu „konkatenacją”. To dość przydatne słowo, z którym zetkniesz się w tej książce jeszcze wielokrotnie.*





## Jazda próbna

Czas odpalić Dyrdymatomat. 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 powiedziec na zebraniu.

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

```
Python 3.6.5 Shell
File Edit Format Run Options Window Help
Python Shell
import random
>>>
czasowniki = ['zorganizować', 'Zresetować',
              'zorganizować', 'Zaadaptować ']

przymiotniki = ['przetestowane', 'freemiumowe',
                'hiperlokalne', 'zsiłoizowane', 'blockchainowe',
                'zorientowane na użytkownika', 'osadzone w chmurze',
                'oparte na API']

rzeczowniki = ['aktywa', 'zasoby',
               'procedury', 'generatory leadów', 'algorytmy',
               'procesy', 'punkty krytyczne', 'paradygmaty']

czasownik = random.choice(czasowniki)
przymiotnik = random.choice(przymiotniki)
rzeczownik = random.choice(rzeczowniki)

zdanie = czasownik + ' ' + przymiotnik + ' ' + rzeczownik
print(zdanie)
```

Zgamifikujemy przetestowane paradygmaty!

Kilkakrotnie odpaliliśmy Dyrdymatomat; oto nasze wyniki.



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:rf59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/LearnToCode/r01/phraseomatic.py =====
Zgamifikować freemiumowe procedury
>>>
===== RESTART: D:/LearnToCode/r01/phraseomatic.py =====
Zdecyfryzować osadzone w chmurze procesy
>>>
===== RESTART: D:/LearnToCode/r01/phraseomatic.py =====
Zdecyfryzować oparte na API procesy
>>>
===== RESTART: D:/LearnToCode/r01/phraseomatic.py =====
Zsynchronizować freemiumowe paradygmaty
>>>
===== RESTART: D:/LearnToCode/r01/phraseomatic.py =====
Zgamifikować freemiumowe generatory leadów
>>>
```

Aby ponownie uruchomić Dyrdymatomat, 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 zamięłowania 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 wyjściowymi w powłoce Pythona.



## Zaostż 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 Założ robaka na haczyk.
- 2 Zarzuć przynętę do wody.
- 3 Przyglądaj się sptawikowi, dopóki się nie zanurzy.
- 4 Wyciągnij rybę.
- 5 Jeśli skończyłeś wędkować, idź do domu, w przeciwnym razie wróć do punktu 1



## Zaostż 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?

- |  |  |
|--|--|
| <p><input checked="" type="checkbox"/> A. Jeżeli w wodzie nie ma ryb, to będziesz zajmował się wędkowaniem przez bardzo długi czas (mniej więcej całą wieczność).</p> <p><input checked="" type="checkbox"/> B. Nie weźmiesz pod uwagę tego, że robak mógłby zlecieć z haczyka i że będziesz musiał założyć nowego.</p> <p><input checked="" type="checkbox"/> C. Co się stanie, jeśli robaki się skończą?</p> | <p><input checked="" type="checkbox"/> D. Czy wskazaliśmy, co zrobić z rybą po wyciągnięciu jej z wody?</p> <p><input checked="" type="checkbox"/> E. W tym przepisie nie ma ani jednego słowa o wędce.</p> <p><input checked="" type="checkbox"/> F. <i>Czy wiadomo konkretnie, jak wygląda dobre zarzucenie? Czy trzeba ponownie zarzucić wędkę, jeśli haczyk wylądje na liliu wodnej?</i><br/><i>Kiedy sptawik zanurza się pod wodą, na ogół trzeba wykonać zacięcie, zanim zacznie się wyciągać rybę. W przepisie nie ma o tym mowy.</i><br/><i>Skąd mamy wiedzieć, że skończyliśmy wędkować? Po godzinie na zegarku? Po tym, że skończyły się robaki? Po czymś innym?</i><br/><i>W przepisie pojawia się wiele założeń. Zapewne przyszło Ci do głowy wiele innych instrukcji, których nie podano w przepisie.</i></p> |
|--|--|
- Wygląda na to, że „wszystkie powyższe” jest poprawną odpowiedzią.



## Magnesiki kodu. Rozwiązanie

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

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 omelet z serem:

Jeśli chcesz omelet z serem, dodaj go.

Posyp serem

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

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!')
```



# Skorowidz

## A

- abstrakcje, 173
- adres
  - internetowy, 422
  - URL, 423, 424
- agregacja, 549
- aktualizacja pliku, 287
- algorytm, 4, 5, 28
  - sortujący, 219
- analitka danych, 238
- analiza tekstu, 242
- API, application programming interface, 420, 428
  - Open Weather Web, 424
  - Spotify, 424
- aplikacja biznesowa, 21
- argumenty, 186
  - kluczowe, 204, 206
- atrybut, 309, 322, 526, 549
  - self, 512

## B

- bezpośrednie obsługiwane wyjątków, 404
- białe znaki, 26
- biblioteka, 10, 565
  - Beautiful Soup, 565
  - Django, 561, 565
  - Fiask, 561, 565
  - Pillow, 565
  - Pygame, 565
- blok kodu, 103, 112
  - przekształcenie w funkcję, 177, 210
- błąd
  - IndexError, 130
  - invalid syntax, 22
  - SyntaxError, 43
- błędy, 54, 402
  - semantyczne, 55

- składniowe, 54
- uruchomieniowe, 54
- użytkownika, 112

brak cudzysłowu, 26

## C

- ciało funkcji, 181, 192, 210
- ciąg
  - Fibonacciego, 345
  - znaków, 64
- cudzysłowy
  - podwójne, 42
  - pojedyncze, 42
  - potrójne, 275

## D

- dane
  - ciągu Fibonacciego, 347
  - tekstowe, 432
  - wejściowe, 37
- data i godzina, 557
- debugowanie, 54, 56
- decyzje, 71
- defekt, 54
- definiowanie
  - funkcji, 177, 179
  - wzorców, 491
- deklarowanie wymaganych parametrów, 205
- dekoratory, 563
- delegacja, 542
- diagram
  - klasy, 306, 532
  - wywołań funkcji, 377
- dodawanie
  - dziedziczenia, 516
  - list, 153
  - notek dokumentacyjnych, 291, 292
  - obiektu żółwia, 301

- obsługi wyjątków, 406
- widżetów, 469
- wyświetlacza graficznego, 436

dokumentacja, 291

- API, 423
- kodu, 96
- pliku, 293

domyślne wartości parametrów, 204

dostęp do pliku, 411

działania na listach, 153, 154

działanie

- funkcji, 178
- konstruktora, 510
- listy, 124
- metody, 514
- pętli for, 138
- pętli while, 104, 106
- przycisku, 479
- sortowania bąbelkowego, 220
- webowych API, 421

dziedziczenie, 516, 526, 540

- wielokrotne, 523, 549

dzielenie

- kodu, 100
- zadań, 2

## E

- edycja komórek, 486
- edytor, 18
  - IDLE, 18, 28
- eksperymenty z żółwiem, 303, 326
- enkapsulacja, 534
- ewaluacja
  - kodu, 49
  - leniwa, 562
  - rekurencyjnego kodu, 341
  - wartości boolowskich, 114, 115
  - wrażenia, 50
  - wrażenia warunkowego, 105

- F**  
fałsz, 78  
folder, 20  
formułowanie heurystyki, 259  
fraktal, 367, 371  
    Kocha, 370  
framework  
    Django, 565  
    Fiask, 565  
funkcja, 173, 186, 197, 210  
    aktualizująca, 473, 474  
    get, 431  
    import, 76  
    input, 28, 38, 56, 64  
    int, 59, 64, 69  
    isinstance, 549  
    koch, 368  
    len, 127, 275, 351  
    losowania, 24  
    main, 410  
    mainloop, 498  
    obliczająca czytelność, 244  
    open, 385, 411  
    pomocy help, 290  
    print, 28, 41, 61, 70  
    random.choice, 25  
    range, 141, 164, 169  
    rekurencyjna, 332  
    split, 248, 275  
    str, 136  
    sum, 330  
    wczytywania wzorców, 492  
funkcje  
    ciało, 181  
    działanie, 178  
    kodu analizatora, 294  
    kolejność argumentów, 210  
    mieszające, 355  
    parametry, 186  
    pierwszoklasowe, 564  
    przekazywanie argumentów, 186, 200  
    rekurencyjne, 338  
    sortujące, 231  
    tworzenie, 177, 250  
    tworzenie abstrakcji, 184  
    wywołanie, 178, 193, 198  
    zwracanie wyniku, 186
- G**  
generowanie  
    awatarów, 190  
    liczb losowych, 76, 112  
    raportu, 144  
    sekwencji liczb, 141  
gra  
    w kamień, papier i nożyce, 72  
    formułowanie logiki, 92  
    gotowa, 109  
    niepoprawne dane wejściowe, 98  
    sprawdzenie wyboru, 86  
    wskazywanie zwycięzcy, 88  
    wybór komputera, 82, 96  
    wybór użytkownika, 83, 96  
    wyświetlanie zwycięzcy, 94  
w odgadywanie kolorów, 110  
w ping-ponga, 508  
w Szalone historie, 382, 384  
w życie, 450, 454, 498  
    logika kodu, 464  
    rozmieszczanie widżetów, 471  
    stan każdej komórki, 460  
    stan pokolenia, 460  
    zasady, 462, 501  
z żółwiami, 312  
    implementacja, 318  
    kod, 315  
    przygotowanie, 314  
    testowanie kodu, 316, 320  
graficzny interfejs użytkownika, 498  
grafika, 302, 436
- H**  
handler  
    kliknąć, 477  
    OptionsMenu, 489  
    widok\_siatki, 487  
heurystyka, 237, 259, 275  
HTTP, 422
- I**  
IDLE, 18  
ignorowanie konsekwentnych samogłosek, 262  
implementacja  
    funkcji serwisu, 358  
    gry, 318  
    klasy, 517, 532  
    przycisku, 480  
    raportu, 144  
    sortowania bąbelkowego, 226  
    logiki gry, 90  
import  
    ch1text, 251, 272  
    datetime, 557  
    random, 24, 82, 323  
    requests, 444  
    turtle, 298  
indeks  
    czytelności, 239, 270, 271  
    elementu, 123  
    końcowy, 265  
indeksy ujemne, 129  
instalacja pakietu requests, 426  
instancja obiektu, 322  
instancjonowanie, 324, 549  
instrukcja  
    break, 392, 395, 411  
    except, 411  
    finally, 411  
    if, 100, 112  
    import, 25, 112  
    list, 227  
    przypisania, 41  
    random, 76  
    return, 186, 210  
    while, 103, 112  
instrukcje try/except, 411  
interaktywność, 449  
interfejs  
    API, 419  
    obiektu, 549  
    użytkownika, 449  
interpretacja kodu, 13  
interpreter, 225, 234, 262, 278  
iteracja, 107, 121, 343, 344  
    zagnieżdżona, 217  
iterator, 395, 411

- iterowanie
  - listy, 134, 138
  - po słowniku, 352
- J**
- język programowania, 7, 8
  - C, 9
  - Lisp, 9
  - Objective-C, 8
  - PHP, 8
  - semantyka, 7
  - składnia, 7
- języki naturalne, 28
- JSON, 429–435, 446
- K**
- kalkulator, 34
- kamień, 72, 281
- klasa, 306–310, 322, 327, 509
  - pochodna, 526
- klucze, 351
  - w słownikach, 371
- kod, 36
  - generatywny, 498
  - liczący sylaby, 260
  - po stronie serwera, 561
- kodowanie, 6, 28
  - kalkulatora, 34
- komentarze, 97, 112
- komunikacja z weboowymi API, 446
- komunikat o błędzie, 403
- konkatenacja, 25, 46, 61, 64
- konstruktor, 308, 324, 509, 549
  - OptionsMenu, 498
- kontroler, 473
- krojenie łańcuchów, 266
- krotka, 559
- L**
- liczba
  - samogłosek, 261
  - słów, 246
  - sylab, 256, 268
  - zdań, 249
- liczby
  - całkowite, 64
  - Fibonacciego, 362
- losowe, 76
  - zmiennoprzecinkowe, 64
- licznik, 148
- listy, 25, 121, 123, 164
  - dodawanie list, 153
  - działanie, 124
  - element, 123
  - heterogeniczne, 130
  - iterowanie, 134, 138
  - kolejność elementów, 130
  - liczba elementów, 127
  - mutowalne, 255
  - ostatni element, 127
  - pobieranie elementu, 125
  - puste, 130
  - równoległe, 348
  - składane, 556
  - tworzenie, 152
  - usuwanie elementu, 153
  - wstawianie elementu, 154
  - wyszukiwanie, 156, 158
  - zmiana wartości, 125
- logika gry, 90, 92, 96
- losowanie, 24
- Ł**
- łańcuchy, 42, 56, 64, 237, 254, 310
  - niemutowalne, 255
  - puste, 112
  - zero, 405, 417
- M**
- memoizacja, 365, 371
- menedżer układu siatki, 470
- menu Run, 27
- metoda, 281, 307, 324, 327, 526
  - after, 483
  - append, 153, 164
  - close, 411
  - extend, 153, 164
  - get, 446
  - insert, 164
  - isinstance, 549
  - read, 411
  - readline, 393, 395, 411
  - strip, 399, 400, 411
- metody
  - nadpisywanie, 524
  - rozszerzanie, 524
- model danych, 459
- moduł, 10, 64, 281–284, 324
  - logging, 565
  - analize, 293
  - datetime, 295, 557
  - json, 435, 446
  - plik analyze.py, 289
  - random, 24, 77, 112
  - requests, 431
  - sched, 565
  - sys, 411
  - Tkinter, 498
  - turtle, 296, 324
- mutowalność, 275
- myślenie komputacyjne, 1
- N**
- nadklasa, 526, 549
- nadpisywanie metod, 524
- narzędzie pip, 446
- nawias, 100
  - kwadratowy, 25, 123
- nazwa
  - pliku, 20
  - zmiennej, 41, 51
- niepoprawne dane wejściowe, 98
- nieporządkowy typ danych, 350
- notacja
  - literałowa, 352
  - obiektowa, 429
- nożyce, 72
- O**
- obiekt, 281, 305–310, 322, 549
  - ekranu, 437
  - odpowiedzi, 432
  - plikowy, 389
- obiekty modułu turtle, 324
- obliczanie, 48
  - indeksu, 240
  - liczby sylab, 256
  - liczby zdań, 249
  - stanu komórki, 460

## Skorowidz

obliczanie  
  stanu pokolenia, 460  
  sumy, 330  
obsługa  
  niepoprawnych danych, 98  
  wyjątków, 403, 406  
odczyt z pliku, 385, 389  
odczytywanie  
  szablonu, 383  
  tekstu, 385  
odpowiedź, 421, 432  
odwzorowanie siatki, 459  
okno edytora, 18  
Open Notify, 433, 441  
operator, 47  
  in, 252  
  konkatenacji, 64  
  logiczny, 91  
  and, 90, 112  
  or, 90, 112  
  not, 90, 91, 112  
modulo, 47  
negacji, 47  
relacji, 78, 91  
  ==, 112  
  >, 112  
  <, 78, 112  
ostatni element listy, 128

## P

pakiet requests, 425, 446, 565  
palindrom, 166, 335–340  
papier, 72  
parametry, 186, 196  
  wymagane, 205  
  zaawansowane, 204  
PEMDAS, 48  
pętla, 71, 103  
  for, 103, 138, 147, 164  
  działanie, 138  
  zakres liczb, 141  
  while, 103–106, 147, 164, 392  
pętle  
  nieskończone, 111  
  zagnieżdżone, 230  
pierwszeństwo operatorów, 47, 48

pierwszy program, 21  
pisanie kodu, 6, 13  
planowanie gry, 313  
plik `analyzer.py`, 289  
pliki  
  odczyt, 385, 389  
  przeszukiwanie, 390  
  wczytywanie do kodu, 389  
  zapisywanie, 379  
  zwracanie, 379  
pobieranie  
  danych, 383  
  danych wejściowych, 37  
  elementu listy, 125  
podejmowanie decyzji, 80  
podklasy, 518, 522, 526, 549  
podłańcuchy, 247, 278  
polecenie  
  File/New File, 18, 26  
  File/Save, 20  
  print, drukuj, 22  
  Run/Run Module, 21, 22, 27  
polimorfizm, 537  
położenie stacji, 443  
pomoc, 290  
porządkowanie danych, 217, 221  
potęgowanie, 47  
prawda, 78  
proces kodowania, 6  
procesor tekstu, 19  
program główny, 288  
programowanie  
  obiektywne, 505, 507, 549  
  reaktywne, 498  
  sterowane zdarzeniami, 476, 478, 498  
projekt Motel dla Psiaków, 531  
projektowanie gry, 74  
proste wartości, 33  
protokół HTTP, 422  
prototypowanie na papierze, 498  
prymitywy, 121  
przechowywanie wartości, 38  
przekazywanie wartości funkcjom, 200  
przekształcanie  
  pseudokodu, 36, 66  
  w łańcuch, 136

przepis, 3, 4  
przeszukiwanie plików, 390  
przetwarzanie tekstu szablonu, 397  
przewidywanie błędów, 112  
przycisk, 479  
przypisanie wartości zmiennej, 39, 44  
pseudokod, 4, 35  
  kalkulatora, 36, 52  
  sortowania bąbelkowego, 223  
  tworzący raport, 132  
pusty łańcuch, 395, 411  
Python, 10  
Python 1.0, 15  
Python 2.0, 15  
Python 3.0, 15

## R

raport  
  generowanie, 144  
  implementacja, 144  
  o roztworach, 145  
  o wynikach testów, 132  
  testowanie, 145, 157  
refaktoryzacja kodu, 188, 193  
rekurencja, 329, 333, 343, 371  
rekurencyjna funkcja, 368  
rekurencyjne wykrywanie  
  palindromów, 336  
relacja, 520  
  HAS-A, 528  
  IS-A, 549  
requests, 425  
rodzaje błędów, 54, 64  
rozszerzanie metod, 524  
rozszerzenie `.py`, 20

## S

samogłoski, 261  
schemat blokowy, 74  
sekwencja  
  liczb, 141  
  ucieczki, 411  
  znaków, 254  
semantyka języka, 7  
separator, 386  
serwis społecznościowy, 348

## siatka

- rozieszczanie widżetów, 471
- symulatora, 459
- sklejanie tekstu, 25
- składnia języka, 7, 64
- słownik, 329, 350, 371
  - iterowanie, 352
  - użytkowników, 358
  - w słowniku, 357
  - wydajność, 355
  - wykorzystanie, 354
  - zapisywanie liczb Fibonacciego, 364
- słowo kluczowe, 51, 64
  - class, 512
  - def, 177
  - elif, 81, 112
  - else, 80, 112
  - global, 201, 210
  - if, 80
  - while, 103
- sortowanie, 217
  - bąbelkowe, 220–224, 230
- sprawdzanie
  - relacji, 520
  - wyboru użytkownika, 86
- stos wywołań, 339–342
- stosowanie
  - dziedziczenia, 540
  - polimorfizmu, 538
- struktury danych, 121, 123
- styl programowania, 476
- sumowanie listy
  - przypadek bazowy, 331
  - przypadek rekurencyjny, 331
- symulator gry, 454, 499
- systemy generatywne, 452
- szablon, 396
  - przetwarzanie tekstu, 397
- szkielet funkcji, 250

## Ś

- ścieżki
  - bezwzględne, 387
  - względne, 386
- śledzenie ISS, 445

## T

- tablice asocjacyjne, 329
- tabulator, 394
  - pionowy, 394
- tekst, 237
  - liczba słów, 245, 246
  - liczba sylab, 256
  - liczba zdań, 249
  - usuwanie końcowych liter, 264
  - wielowierszowy, 242, 275
- testowanie kodu, 22, 161, 190, 316, 530, 544
  - funkcji ciągu Fibonacciego, 346
  - gry Szalone historie, 398
  - liczenia sylab, 268
  - metody after, 484
  - Pythona, 19
  - raportu, 145, 157
  - równości, 112
  - Serwisu Aspołecznościowego, 360
  - Szalonych historii, 408
  - układu, 470
  - wyniku, 62
- tworzenie
  - abstrakcji kodu, 173, 184
  - algorytmu, 6
  - folderu, 20
  - funkcji, 177, 250
  - grafiki, 302
  - klasy, 509, 510
  - kodu szkieletowego, 250
  - listy, 25, 152
  - metody, 513
  - modelu danych, 459
  - obiektów, 306, 324, 549
  - podklasy, 518, 522
  - słowników, 352
  - symulatora, 458
  - usługi, 541
  - widoku, 467
  - widżetu, 468
  - zółwia, 296, 298, 304
- typ danych None, 208
- typy danych, 33, 58
  - krotki, 559
  - liczby, 121

## listy, 123

- łańcuchy, 121
- słowniki, 350
- wartości logiczne, 121
- zbiory, 560
- typy numeryczne, 64

## U

- układ
  - siatki, 472
  - współrzędnych, 438
- URL, 422
- uruchamianie
  - kodu, 13, 40
  - programu, 6
  - z wiersza poleceń, 410
- usuwanie
  - elementu z listy, 153
  - końcowych liter, 264
  - spacji, 135
- używanie
  - argumentów kluczowych, 206
  - pseudokodu, 37
  - widżetów, 468

## W

- wartości, 49, 64, 67, 351
  - domyślne, 204
  - logiczne, 71, 78, 79
  - False, 78, 112
  - True, 78, 112
- wartość None, 208
- wcięcie, 26
- wczytywanie
  - pliku, 389
  - szablonu, 396
  - wzorców, 492
- webowe interfejsy API, 419, 446
- wersje Pythona, 16
- widok, 467
- widżety, 449, 451, 467, 498
- wielkość liter, 101
- wielokrotne
  - wykorzystywanie kodu, 176
  - wyświetlanie zapytań, 108

## Skorowidz

- wiersz poleceń, 410
  - właściwości, 324
  - wprowadzanie kodu, 43
  - współrzędne geograficzne, 438
  - wybór
    - losowy, 84
    - niepoprawny użytkownika, 99
  - wyjątki, 402
    - bezpośrednie obsługiwane, 404
    - wyłapywanie, 411
  - wykrywanie błędów, 59
  - wynik, 61
  - wyrażenia, 47, 49, 67
    - logiczne, 112, 118
    - porównujące wartości, 78
    - regularne, 558
    - warunkowe, 105, 112
  - wysyłanie żądań, 425, 433
  - wyszukiwanie
    - kluczy, 371
    - najwyższych wyników, 156
    - użytkownika, 359
  - wyścigi żółwi, 312
  - wyświetlanie
    - ISS, 440
    - łańcucha, 106
    - położenia ISS, 436
    - zapytania, 101, 102, 108
    - zapytania cyklicznego, 102
  - wywołanie
    - funkcji, 64, 178, 193, 198
    - rekurencyjne funkcji, 334
  - wzorce
    - definiowanie, 491
    - predefiniowane, 488
    - wczytywanie, 492
  - wzór na czytelność, 270
- ## Z
- zachowania emergentne, 451
  - zakres liczb, 141, 142
  - zapamiętywanie wyników, 364
  - zapisywanie
    - historyjki, 407
    - liczb Fibonacciego, 364
    - plików, 379
    - programu, 20
  - zapytanie cyklicznie, 102
  - zasięg zmiennej, 196
  - zastosowania zakresów, 142
  - zbiór, 560
    - zadań, 28
    - zdarzenia, 449, 451, 476, 478
      - innego rodzaju, 481
    - zmienianie wartości
      - w liście, 125
      - zmiennej, 44, 46
    - zmienna, 33, 38, 64, 195
      - \_\_name\_\_, 286, 324
    - zmiennie
      - globalne, 196, 201–203, 210
      - instancji, 324
      - lokalne, 188, 194, 202, 210
      - nazwa, 46, 51
      - przypisanie wartości, 44, 46
      - zasięg, 196
    - znaczniki, 397, 399
    - znak
      - =, 112
      - krzyżyka, 24, 97
      - nowego wiersza, 394
      - zachęty, 43
    - znaki ==, 112
    - zwracanie plików, 379

## Ż

    - żądanie, 421, 431, 446
      - do Open Notify, 43



# PROGRAM PARTNERSKI

— GRUPY HELION —

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

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

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

GRUPA  
**Helion** 

# Rusz głową! Nauka programowania



Programista to szczególny typ specjalisty. Jeśli uważasz, że myślisz 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 dziwnie, 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 oczywiście 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
- widżety 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.

	<p>Sprawdź nasze szkolenia!</p>
<p>helion.pl</p> <p>HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl</p>	 <p>AKADEMIA IT &amp; BUSINESS</p> <p>WWW.SZKOLENIA.HELION.PL</p>
<p>INFORMATYKA W NAJLEPSZYM WYDANIU</p>	

KOD KORZYŚCI  
Sięgnij po więcej! ▶



ISBN 978-83-283-4697-0



9 788328 346970

Cena: 99,00 zł