O'REILLY®

Wydanie II

# Python

## Nowoczesne programowanie w prostych krokach

Helion

Bill Lubanovic

Tytuł oryginału: Introducing Python: Modern Computing in Simple Packages, 2nd Edition

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-6842-2



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

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

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

Printed in Poland.

# Spis treści

## Część II. Python w praktyce

ROZDZIAŁ 1.

# Przedsmak Pythona

*Uznanie zyskują tylko okropne języki. Python jest wyjątkiem.*

— Don Knuth

## Zagadki

Zacznijmy od dwóch minizagadek i ich rozwiązań. Jak myślisz, co oznaczają poniższe wiersze?

```
(Rząd 1): (PS) K18, pomiń, k1, odwróć.
(Rząd 2): (LS) Sl 1 dług., p5, 2o. na lewo, p1, odwróć.
```

Zapisy wygląda technicznie, niczym jakiś program komputerowy. W rzeczywistości jest to *wzór dziewiarski* na piętę skarpety, takiej jak na rysunku 1.1.

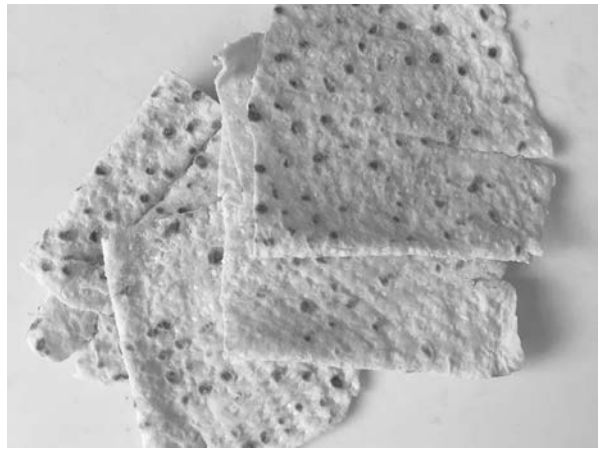
*Rysunek 1.1. Skarpety zrobione na drutach*

Dla mnie ten wzór jest równie zrozumiały jak łamigłówka sudoku dla moich kotów. Natomiast dla mojej żony jest oczywisty. Dla Ciebie pewnie też, jeżeli lubisz robótki ręczne.

Przyjrzyjmy się innemu tajemniczemu tekstowi — liście składników. Jej przeznaczenie jest zupełnie jasne, choć nie wiadomo, jaki jest końcowy produkt.

```
1/2 k. masła lub margaryny
1/2 szk. kremu
2 1/2 szk. mąki
1 mała ł. soli
1 duża ł. cukru
4 szk. startych ziemniaków (zimnych)
Wszystkie składniki przed zmieszaniem z mąką muszą być zimne.
Wymieszaj wszystkie składniki.
Dokładnie ugnieć.
Uformuj 20 kulek i umieść je w chłodnym miejscu.
Dla każdej kulki:
  Rozsyp mąkę na szmatce.
  Rozwałkuj kulkę na okrągły placek.
  Smaż na patelni do uzyskania brązowego koloru.
  Odwróć i smaż na drugiej stronie.
```

Nawet jeżeli nie gotujesz, poznasz, że jest to przepis kulinarny[1]. Składa się z listy składników i wskazówek do wykonania. Ale na co to jest przepis? Na lefse, norweski specjał, podobny do tortilli (patrz rysunek 1.2). Posmaruj go masłem, dżemem lub czymkolwiek innym, zwiń i smacznego!



*Rysunek 1.2. Lefse*

Wzór dziewiarski i przepis kulinarny mają kilka wspólnych cech:

- Ustalony **zbiór** słów, skrótów i symboli. Niektóre są zrozumiałe, inne brzmią tajemniczo.
- **Składnię**, czyli zasady określające sposób wyrażania treści.
- **Sekwencję operacji**, które należy wykonać w zadanej kolejności.

[1] Zazwyczaj można go znaleźć w książkach kucharskich lub tajemnych zapiskach kucharzy.

- Sekwencje powtarzanych operacji (**pętle**), na przykład sposób smażenia placka lefse.
- Odwołania do innych sekwencji operacji (**funkcji** w terminologii informatycznej). W przepisie kulinarnym może to być odwołanie do przepisu ścierania ziemniaków.
- Założenie dotyczące znajomości **kontekstu**. W przepisie przyjęte jest założenie, że ten, kto go czyta, wie, co to jest woda i jak się ją gotuje. Wzór dziewiarski zakłada, że umiesz robić na drutach i nie kłujesz się nimi zbyt często.
- Wykorzystywane, tworzone i modyfikowane **dane** — tutaj ziemniaki i włóczka.
- **Narzędzia** do przetwarzania danych — garnek, mikser, patelnia, druty.
- Oczekiwany efekt. W opisanych przykładach jest to coś, co można zjeść lub na siebie założyć.

Tego rodzaju przykłady możesz znaleźć wokół siebie wszędzie. Są to idiomy, żargony lub specjalne języki. Ludziom, którzy je znają, pozwalają oszczędzić czas, ale dla innych brzmią tajemniczo. Spróbuj zrozumieć artykuł o brydżu, jeżeli w niego nie grasz, lub specjalistyczną publikację, jeżeli nie jesteś naukowcem (lub nim jesteś, tylko w innej dziedzinie).

# Kilka małych programów

Wszystkie opisane w poprzednim podrozdziale elementy znajdziesz w programach komputerowych, napisanych w specyficznych językach, za pomocą których ludzie mówią komputerom, co mają robić. Użyłem przykładów ze wzorem dziewiarskim i przepisem kulinarnym, aby pokazać, że w programowaniu nie ma niczego tajemniczego. Opanowanie tej sztuki polega głównie na zapamiętaniu pewnych słów i zasad ich stosowania.

Jest dobrze, jeżeli nie ma zbyt wielu słów i zasad, których trzeba się nauczyć. Ludzki mózg jest w stanie pomieścić tylko określoną ilość informacji.

Zajmijmy się wreszcie jakimś prostym programem (patrz listing 1.1). Jak myślisz, co on robi?

*Listing 1.1. Program odliczanie.py*

```
for countdown in 5, 4, 3, 2, 1, "Cześć!":
    print(countdown)
```

Jeżeli domyśliłeś się, że jest to program w języku Python wyświetlający następujące wiersze:

```
5
4
3
2
1
Cześć!
```

przyznasz, że łatwiej jest zrozumieć kod napisany w tym języku niż przepis kulinarny czy wzór dziewiarski. Poza tym programy w Pythonie możesz pisać w komfortowym i bezpiecznym zaciszu swojego gabinetu, z dala od gorących garnków i ostrych przedmiotów.

Programowanie w Pythonie polega na stosowaniu specjalnych słów i symboli (`for`, `in`, `print`, przecinków, średników, nawiasów), będących ważnymi elementami składni (zasad) tego języka. Dobra wiadomość jest taka, że składnia Pythona jest przyjemniejsza i łatwiejsza do opanowania niż w innych językach. Kod napisany w tym języku brzmi całkiem naturalnie — prawie jak przepis kulinarny.

Listing 1.2 przedstawia inny program, który wybiera z listy jedno z zaklęć Harry'ego Pottera i wyświetla je na ekranie.

*Listing 1.2. Program zaklęcia.py*

```
spells = [
    "Riddikulus!",
    "Wingardium Leviosa!",
    "Avada Kedavra!",
    "Expecto Patronum!",
    "Nox!",
    "Lumos!",
    ]
print(spells[3])
```

Poszczególne zaklęcia są **ciągami znaków** (sekwencjami znaków ujętymi w cudzysłowy). Rozdzielone przecinkami i ujęte w nawiasy klamrowe ([ i ]), tworzą **listę**. Słowo `spells` jest **zmienną**, czyli nazwą, za pomocą której można z listą robić różne rzeczy. W tym przykładzie program wyświetla czwarte zaklęcie.

```
Expecto Patronum!
```

Dlaczego do wyświetlenia czwartego zaklęcia została użyta cyfra 3? Lista w Pythonie jest sekwencją wartości, do których można odwoływać się za pomocą **przesunięcia** względem początku listy. Pierwsza wartość ma przesunięcie równe 0, a czwarta równe 3.

Ludzie odliczają od 1, zatem odliczanie od zera wygląda przedziwnie. Dlatego lepiej jest używać pojęcia *przesunięcie* zamiast *pozycja*. Jest to jeden z przykładów różnic pomiędzy językiem programowania a językiem ludzi.

Lista jest bardzo popularną **strukturą danych** w Pythonie. W rozdziale 7. dowiesz się, jak ją stosować.

Program przedstawiony w listingu 1.3 wyświetla jeden z cytatów z „Kubusia Puchatka". Tym razem jednak użyte jest odwołanie wykorzystujące nazwę bohatera, a nie pozycję w liście.

*Listing 1.3. Program cytaty.py*

```
quotes = {
    "Puchatek": "Jestem Misiem o Bardzo Małym Rozumku",
    "Kłapouchy": "Wypadek to bardzo dziwna rzecz. Nigdy go nie ma, dopóki się nie wydarzy.",
    "Tygrysek": "Jest to tak skomplikowane, że nawet ja to rozumiem.",
    }
character = "Tygrysek"
print(character, "powiedział:", quotes[character])
```

Po uruchomieniu tego prostego programu na ekranie pojawi się:

```
Tygrysek powiedział: Jest to tak skomplikowane, że nawet ja to rozumiem.
```

Słowo `quotes` jest zmienną, w tym przypadku nazwą **słownika**. Słownik to kolekcja unikatowych kluczy (tutaj imion bohaterów) i skojarzonych z nimi wartości (cytatów). Często jest to bardziej przydatna struktura niż lista, ponieważ umożliwia odwoływanie się do wartości za pomocą nazw.

W listingu 1.2 zostały użyte nawiasy kwadratowe ([ i ]) do zdefiniowana listy, a w listingu 1.3 nawiasy klamrowe ({ i }) do zdefiniowania słownika. Klucze z wartościami w słowniku są kojarzone za pomocą dwukropka (`:`). Więcej o słownikach dowiesz się w rozdziale 8.

Mam nadzieję, że nie była to dla Ciebie zbyt duża dawka informacji o składni. W kolejnych rozdziałach po trochu będziesz dowiadywał się coraz więcej o obowiązujących prostych regułach.

# Większy program

Teraz zajmiemy się czymś zupełnie innym. Listing 1.4 przedstawia program wykonujący serię dość skomplikowanych operacji. Na razie nie staraj się zrozumieć, jak ten program działa — ta książka jest od tego, abyś w odpowiednim czasie się tego dowiedział. Celem przykładu jest pokazanie Ci, jak wygląda typowy niebanalny program w Pythonie. Jeżeli znasz inne języki programowania, możesz je porównać z Pythonem. Nie znasz wprawdzie jeszcze tego języka, ale czy bez zaglądania do opisu niżej potrafisz dociec, co robią poszczególne wiersze? Poznałeś już przykłady użycia listy i słownika. Tutaj pojawia się kilka nowości.

*Listing 1.4. Program archive.py*

```
import webbrowser
import json
from urllib.request import urlopen

print("Poszukajmy jakiejś starej strony.")
site = input("Podaj adres URL strony: ")
era = input("Pdaj rok, miesiąc i dzień, np. 20150613: ")
url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
response = urlopen(url)
contents = response.read()
text = contents.decode("utf-8")
data = json.loads(text)
try:
    old_site = data["archived_snapshots"]["closest"]["url"]
    print("Adres znalezionej kopii: ", old_site)
    print("Za chwilę strona powinna pojawić się w przeglądarce.")
    webbrowser.open(old_site)
except:
    print("Niestety, tej strony nie ma w archiwum.")
```

Przykładowy program opisany w poprzednim wydaniu książki łączył się z witryną YouTube. Pobierał informacje o najwyżej ocenianych filmach i robił to dobrze. Ledwie jednak wyschła farba drugiego wydania, Google zablokowało wykorzystywaną w programie funkcję, więc przestał on działać. Nowy przykład, przedstawiony w listingu 1.4, odwołuje się do innej witryny, dzięki której będzie działał nieco dłużej: Internet Archive (*http://archive.org*). Jest to bezpłatny serwis, w którym przechowywane są miliardy stron (jak również filmy, muzyka, gry i inne cyfrowe starocie), które pojawiły się w Internecie w ciągu ostatnich 20 lat. Więcej przykładów użycia interfejsu API udostępnianego w tej witrynie poznasz w rozdziale 18.

Program prosi użytkownika o podanie adresu URL strony i daty. Następnie wysyła do witryny Internet Archive zapytanie, czy jest dostępna kopia żądanej strony z zadanego okresu. Jeżeli tak, program wyświetla jej adres w archiwum i otwiera ją w przeglądarce. Celem przykładu jest pokazanie, jak za pomocą Pythona można wykonywać różnego rodzaju operacje, takie jak pobieranie danych od użytkownika, przesyłanie ich przez Internet, odczytywanie odebranych treści, wyodrębnianie adresów URL i wyświetlanie stron w przeglądarce.

Po odebraniu z Internetu strony zapisanej w formacie HTML trzeba ją w jakiś sposób wyświetlić. Jest z tym mnóstwo roboty, którą spokojnie można scedować na przeglądarkę. Można też próbować wyodrębnić z kodu potrzebne dane (więcej informacji o surfowaniu po Internecie znajdziesz w rozdziale 18.). Każda z tych opcji wymaga włożenia dużo pracy w napisanie większego programu. Na szczęście witryna Internet Archive zwraca dane w czytelnym dla człowieka formacie JSON (ang. *JavaScript Object Notation* — zapis obiektów JavaScript), opisującym w uporządkowany sposób typy danych i ich zawartość. Format ten jest kolejnym językiem powszechnie stosowanym do przesyłania informacji pomiędzy różnymi komputerami i systemami. Więcej na jego temat dowiesz się w rozdziale 12.

W Pythonie można przekształcać dane JSON w struktury typowe dla tego języka. Tego rodzaju operacje poznasz w dalszych rozdziałach. Ten prosty program wyodrębnia tylko część danych — adres URL strony zapisanej w archiwum. Niemniej jednak jest to kompletny program, który możesz uruchomić. Aby kod był prosty, zaimplementowany w nim został tylko podstawowy mechanizm obsługi błędów. Numery wierszy nie stanowią części programu — są wykorzystywane jedynie w umieszczonym pod nim opisie.

Ten prosty, składający się z kilkunastu wierszy program robi bardzo wiele rzeczy. Użyte w nim instrukcje mogą nie być jeszcze dla Ciebie zrozumiałe, ale poznasz je w najbliższych rozdziałach. Poniżej znajduje się opis operacji wykonywanych w poszczególnych wierszach.

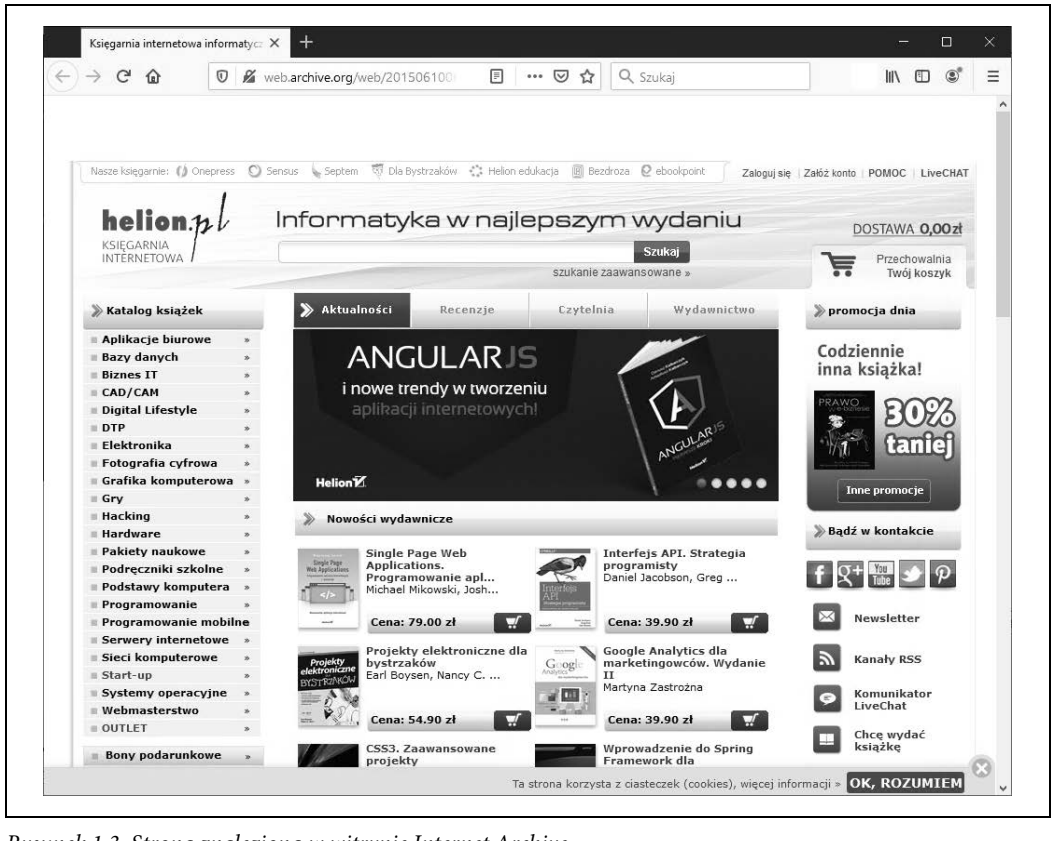
1. *Importowanie* (udostępnianie w programie) całego kodu modułu `webbrowser` zawartego w standardowej bibliotece Pythona.
2. Importowanie całego kodu modułu `json` zawartego w standardowej bibliotece Pythona.
3. Importowanie wyłącznie funkcji `urlopen()` znajdującej się w module `urllib.request`, zawartym w standardowej bibliotece Pythona.
4. Pusty wiersz, aby program był bardziej czytelny.
5. Wyświetlenie powitalnego tekstu na ekranie.
6. Wyświetlenie prośby o podanie adresu URL, odczytanie danych wpisanych przez użytkownika i zapisanie ich w zmiennej o nazwie `site`.
7. Wyświetlenie następnej prośby, o podanie roku, miesiąca i dnia, a następnie zapisanie wpisanych danych w zmiennej o nazwie `era`.
8. Utworzenie zmiennej o nazwie `url` i umieszczenie w niej danych, których witryna Internet Archive potrzebuje do wyszukania podanej strony.
9. Nawiązanie połączenia z usługą i zażądanie informacji o podanym adresie URL.
10. Odebranie odpowiedzi i przypisanie jej do zmiennej.
11. Zdekodowanie treści zapisanej w formacie JSON i przypisanie jej do zmiennej.
12. Przekształcenie tekstu na dane (strukturę).
13. Obsługa błędów: jeżeli podczas wykonywania kolejnych czterech wierszy wystąpi błąd, nastąpi przejście do ostatniego wiersza programu (poniżej instrukcji `except`).
14. Jeżeli zostanie znaleziona strona z zadanego okresu, nastąpi wyodrębnienie jej adresu z trójpoziomowego słownika. Zwróć uwagę, że ten wiersz i dwa następne są wcięte. W ten sposób oznacza się wiersze, które są objęte instrukcją `try`.

15. Wyświetlenie adresu URL żądanej strony w archiwum.
16. Wyświetlenie informacji o tym, co się za chwilę stanie.
17. Otwarcie w przeglądarce strony o znalezionym adresie URL.
18. Jeżeli podczas wykonywania czterech poprzednich wierszy wystąpi błąd, nastąpi przejście do tego wiersza.
19. Wyświetlenie komunikatu, że żądana strona nie została znaleziona. Wiersz jest wcięty, ponieważ ma być wykonany tylko wtedy, gdy zostanie wykonana poprzedzająca go instrukcja `except`.

Po uruchomieniu programu oraz wpisaniu adresu URL i daty pojawi się pokazany niżej przykładowy wynik:

```
$ python archive.py
Poszukajmy jakiejś starej strony.
Podaj adres URL strony: helion.pl
Pdaj rok, miesiąc i dzień, np. 20150613: 20150613
Adres znalezionej kopii:  http://web.archive.org/web/20150610064343/http://helion.pl/
Za chwilę strona powinna pojawić się w przeglądarce.
```

Rysunek 1.3. przedstawia znalezioną stronę wyświetloną w przeglądarce.




*Rysunek 1.3. Strona znaleziona w witrynie Internet Archive*

W opisanym przykładzie wykorzystanych jest kilka modułów (programów zainstalowanych razem z Pythonem) ze **standardowej biblioteki**. Jednak nie jesteś skazany na korzystanie wyłącznie z nich. Dostępnych jest wiele doskonałych zewnętrznych modułów. Listing 1.5 przedstawia inną wersję programu, w której wykorzystany jest doskonały zewnętrzny pakiet `requests`.

*Listing 1.5. Program archive2.py*

```
import webbrowser
import requests

print("Poszukajmy jakiejś starej strony.")
site = input("Podaj adres URL strony: ")
era = input("Podaj rok, miesiąc i dzień, np. 20150613: ")
url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
response = requests.get(url)
data = response.json()
try:
    old_site = data["archived_snapshots"]["closest"]["url"]
    print("Adres znalezionej kopii: ", old_site)
    print("Za chwilę strona powinna pojawić się w przeglądarce.")
    webbrowser.open(old_site)
except:
    print("Niestety, tej strony nie ma w archiwum.")
```

Nowa wersja programu jest krótsza i dla większości programistów bardziej czytelna. Więcej o module `requests` dowiesz się w rozdziale 18., a o zewnętrznych modułach w ogólności — w rozdziale 11.

# Python w praktyce

Czy warto poświęcać czas i siły na uczenie się Pythona? Ten język istnieje od 1991 r. (jest starszy od Javy, ale młodszy od C) i wciąż należy do pięciu najbardziej popularnych języków programowania. Programiści zarabiają pieniądze, tworząc w tym języku poważne serwisy, takie jak Google, YouTube, Instagram, Netflix czy Hulu. Sam używam go do tworzenia produkcyjnych aplikacji wykorzystywanych w różnych dziedzinach. Python ma opinię wydajnego języka, atrakcyjnego dla szybko rozwijających się firm.

Python jest stosowany w różnych obszarach, m.in.:

- wierszach poleceń i terminalach,
- interfejsach graficznych,
- serwisach internetowych po stronie serwera i klienta,
- serwerach wspomagających duże, popularne witryny internetowe,
- chmurach (centrach danych zarządzanych przez zewnętrzne podmioty),
- urządzeniach przenośnych,
- systemach wbudowanych.

Spektrum tworzonych w tym języku programów jest bardzo szerokie — od kilkuwierszowych skryptów, takich jak powyższe przykłady, do potężnych systemów złożonych z milionów linii kodu.

Na stronie *https://oreil.ly/8vK7y* dostępny jest raport Python Developers Survey 2018, zawierający wiele liczb i wykresów pokazujących miejsce Pythona w dzisiejszej informatyce.

Z tej książki dowiesz się, jak ten język jest wykorzystywany w witrynach internetowych oraz w systemach przetwarzania danych i zarządzania nimi. W końcowych rozdziałach opisane są zastosowania Pythona w sztuce, nauce i biznesie.

# Python i inne języki

Jak wypada Python w porównaniu z innymi językami? Gdzie i kiedy należy go stosować? W tym podrozdziale przedstawię przykłady napisane w innych językach, abyś miał wyobrażenie, jak wygląda konkurencja. *Nie* wymagam, abyś je rozumiał, szczególnie jeżeli nie miałeś z nimi do czynienia. (Nim dotrzesz do ostatniego przykładu w tej książce, odetchniesz z ulgą, że nie musiałeś pracować z innymi językami).

Każdy z programów wyświetla liczbę i krótką informację o sobie.

Program używany w terminalu lub wierszu poleceń, odczytujący wpisywane przez użytkownika informacje, uruchamiający inne programy i wyświetlający wyniki, jest nazywany **powłoką**. W systemie Windows jest to program `cmd` (*https://pl.wikipedia.org/wiki/Cmd.exe*). Służy on do uruchamiania plików wsadowych z rozszerzeniem *.bat*. W systemach Linux i Unix (w tym macOS) dostępnych jest wiele różnych powłok. Najpopularniejsza z nich to `bash` (*https://www.gnu.org/software/bash*), zwana też `sh`. Możliwości powłoki są dość ograniczone — pozwalają kodować proste programy i zastępować nazwy plików gwiazdką. Polecenia przeznaczone do późniejszego wykorzystania można zapisywać w plikach zwanych **skryptami powłoki**. Być może były to pierwsze programy, z jakimi miałeś do czynienia jako programista. Problem ze skryptami polega na tym, że nie skalują się dobrze i trudno jest napisać program składający się z więcej niż kilkuset wierszy. Poza tym skrypty działają wolniej niż programy napisane w innych językach. Poniższy listing przedstawia prosty skrypt powłoki.

```
#!/bin/sh
language=0
echo "Język nr $language: cześć, jestem powłoką!"
```

Gdy zapiszesz powyższy kod w pliku *test.sh* i uruchomisz go, na ekranie pojawi się następująca informacja:

```
Język 0: cześć, jestem powłoką!
```

Języki wiarusy, takie jak C (*https://pl.wikipedia.org/wiki/C_(język_programowania)*) i C++ (*https://pl.wikipedia.org/wiki/C%2B%2B*), to języki niskopoziomowe, których używa się wtedy, gdy ważna jest szybkość programu. Twój system operacyjny i spora liczba dostarczanych z nim programów (włącznie z interpreterem Pythona) prawdopodobnie są napisane w jednym z tych języków.

Języki C i C++ są dość skomplikowane, a napisane w nich programy — trudne w utrzymaniu. Kodując w nich, trzeba pamiętać o wielu szczegółach, m.in. o **zarządzaniu pamięcią**. Pomyłki skutkują trudnymi w diagnozowaniu awariami i błędami. Poniżej przedstawiony jest prosty program napisany w języku C:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int language = 1;
    printf("Język nr %d: cześć, jestem C!", language);
    return 0;
}
```

Język C++ jest podobny do C, ale ma kilka bardziej zaawansowanych funkcjonalności. Poniżej przedstawiony jest przykładowy kod.

```
using namespace std;
int main() {
    int language = 2;
    cout << "Język nr " << language << ": cześć jestem C++!";
    return(0);
}
```

Java (*https://www.java.com*) i C# (*https://oreil.ly/1wo5A*) to następcy C i C++, wolni od wielu mankamentów swoich poprzedników, m.in. skomplikowanego zarządzania pamięcią. Tworzone programy są jednak dość rozbudowane. Przykład kodu napisanego w języku Java wygląda tak:

```
public class Anecdote {
    public static void main (String[] args) {
        int language = 3;
        System.out.format("Język nr %d: cześć, jestem Java!", language);
    }
}
```

Jeżeli nie programowałeś w żadnym z tych języków, możesz się zastanawiać, do czego to wszystko jest potrzebne w programie, który wyświetla jeden wiersz tekstu. W niektórych językach narzut składniowy jest bardzo duży. Więcej na ten temat dowiesz się w rozdziale 2.

Języki C, C++ i Java to przykłady **języków statycznych**, które wymagają określania pewnych niskopoziomowych szczegółów, takich jak typy danych. W dodatku A dowiesz się, że na przykład liczba całkowita zajmuje w pamięci komputera określoną liczbę bitów i na danych tego rodzaju można wykonywać tylko operacje arytmetyczne. Przeciwieństwem są **języki dynamiczne** (zwane **językami skryptowymi**), w których nie jest wymagane deklarowanie typów zmiennych przed ich użyciem.

Przez wiele lat popularnym, uniwersalnym językiem dynamicznym był Perl (*http://www.perl.org*). Jest to potężny język, wyposażony w wiele przydatnych bibliotek. Jego składnia jest jednak dość kłopotliwa i dlatego w ostatnich latach stracił uznanie na rzecz języków Python i Ruby. Oto jak wygląda nasz przykład napisany w Perlu:

```
my $language = 4;
print "Język nr $language: cześć jestem Perl!";
```

Bardziej nowoczesnym językiem jest Ruby (*http://www.ruby-lang.org*), który odziedziczył kilka cech po Perlu i zdobył popularność dzięki platformie Ruby on Rails, przeznaczonej do tworzenia stron internetowych. Jest wykorzystywany w tych samych obszarach co Python. Wybór między tymi dwoma językami zazwyczaj jest kwestią gustu i dostępności bibliotek ułatwiających stworzenie potrzebnej aplikacji. Poniżej przedstawiony jest przykład napisany w języku Ruby:

```
language = 5
puts "Język nr #{language}: cześć, jestem Ruby!"
```

Użyty w poniższym przykładzie język PHP (*http://www.php.net*) jest bardzo popularny w dziedzinie stron internetowych, ponieważ można go łatwo łączyć z kodem HTML. Sam w sobie ma jednak kilka niedostatków i poza Internetem nie ma wielu zastosowań. Przykład jego użycia wygląda jak niżej:

```
<?PHP
$language = 6;
echo "Język nr $language: cześć, jestem PHP!";
?>
```

W ostatnim czasie pojawił się język Go (lub Golang, gdybyś chciał poszukać o nim informacji w Google), sprawiający wrażenie języka wydajnego i przyjaznego (*https://golang.org*):

```
package main
import "fmt"
func main() {
  language := 7
  fmt.Printf("Język nr %d: cześć, jestem Go!", language)
}
```

Współczesną alternatywą dla C i C++ jest Rust (*https://doc.rust-lang.org*):

```
fn main() {
    let language = 8;
    println!("Język nr {}: cześć, jestem Rust!", language)
}
```

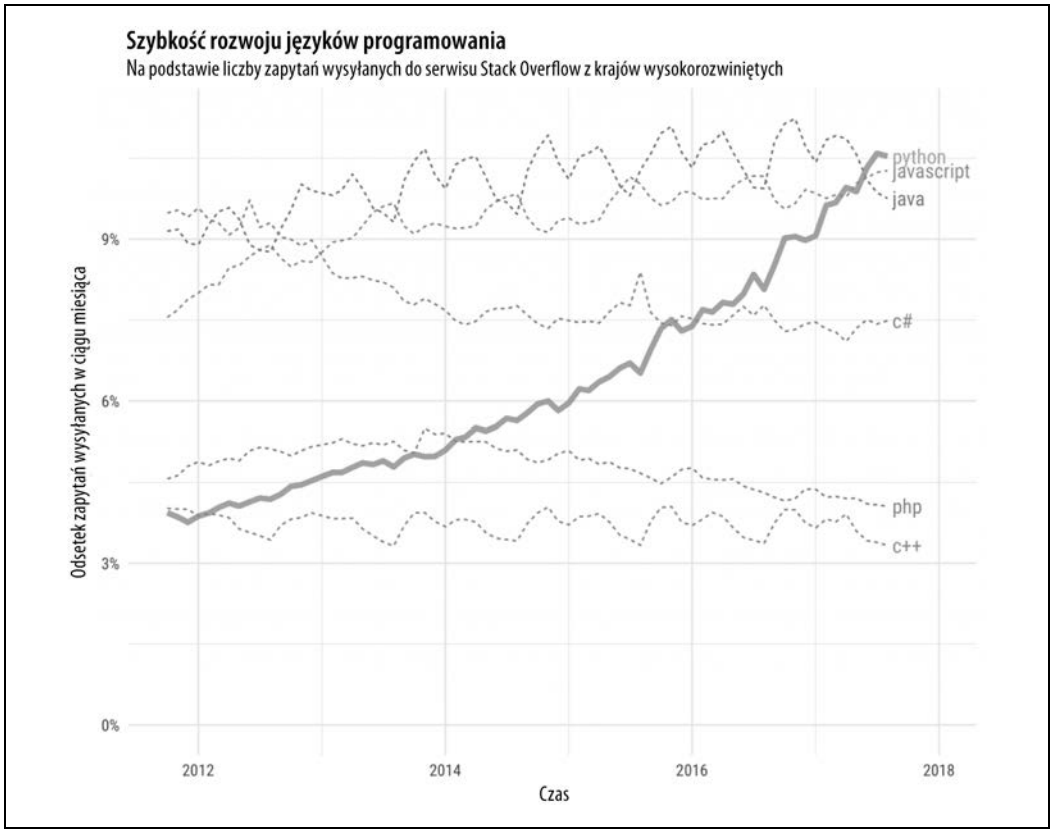
Co jeszcze nam zostało? Ależ tak, Python (*https://python.org*):

```
language = 9
print(f"Język nr {language}: cześć, jestem Python!")
```

# Dlaczego właśnie Python?

Jednym, choć nie najważniejszym, powodem, dla którego warto używać Pythona, jest jego popularność. Według różnych źródeł jest to:

- Jeden z najszybciej rozwijających się języków programowania (*https://oreil.ly/YHqqD*), co ilustruje rysunek 1.4.
- Najbardziej rozwojowy język programowania. W czerwcu 2019 r. autorzy indeksu TIOBE (*https://www.tiobe.com/tiobe-index*) podali, że „w tym miesiącu indeks Pythona pobił rekord wszech czasów i osiągnął wartość 8,5%. Jeżeli ten trend się utrzyma, Python w ciągu 3 – 4 lat może zastąpić języki C i Java i stać się najpopularniejszym językiem programowania na świecie”.




*Rysunek 1.4. Pod względem szybkości rozwoju Python jest liderem wśród języków programowania*

- Najpopularniejszy język programowania w 2018 r. według serwisów TIOBE, IEEE Spectrum (*https://oreil.ly/saRgb*) i PyPL (*http://pypl.github.io/PYPL.html*).
- Najczęściej wybierany język na wykładach z podstaw programowania na największych amerykańskich uczelniach (*http://bit.ly/popular-py*).
- Oficjalnie zatwierdzony w szkołach wyższych we Francji język do nauki programowania.

W ostatnich latach Python zyskał ogromną popularność w dziedzinach badania danych i uczenia maszynowego. Jeżeli chcesz zdobyć dobrze płatną pracę programisty w ciekawej branży, Python obecnie wydaje się najlepszym wyborem. Jeżeli jesteś pracodawcą, możesz wybierać spośród rosnącej liczby doświadczonych programistów.

*Dlaczego* ten język jest taki popularny? Czy są jakieś ukryte powody?

Python jest uniwersalnym, wysokopoziomowym językiem programowania. Dzięki swojej strukturze jest bardzo *czytelny*, co jest ważniejsze, niż się na pozór wydaje. Każdy program komputerowy pisany jest tylko raz, ale czytany i weryfikowany wielokrotnie, często przez różne osoby. Czytelność języka decyduje o łatwości nauczenia się go i zapamiętania. Krzywa uczenia Pythona w porównaniu z innymi językami jest łagodna, co oznacza, że szybko można go zacząć skutecznie wykorzystywać, ale głębokie poznanie wymaga czasu i doświadczenia.

Dzięki zwięzłości Pythona programy są krótsze od napisanych w innych językach. Badania wykazały, że programiści, niezależnie od używanego języka, piszą dziennie mniej więcej stałą liczbę wierszy kodu. Zatem im mniej wierszy musi użyć programista do osiągnięcia jakiegoś celu, tym większa jest jego wydajność. Dla wielu firm, w których jest to ważne, Python jest (niezbyt) tajną bronią.

Poza tym Python jest darmowy. Można w nim pisać wszystko i wszędzie bezpłatnie.

Python działa wszędzie, ponieważ w swojej standardowej bibliotece zawiera mnóstwo przydatnego oprogramowania. W tej książce zaprezentowanych jest wiele przykładów, w których wykorzystana jest standardowa biblioteka oraz liczne użyteczne biblioteki zewnętrzne.

Jednak najważniejszy powód, dla którego warto używać Pythona, jest nietypowy: programiści go lubią i nie uważają stosowania go za zło konieczne do osiągnięcia zamierzonego celu. O tym języku mówi się, że „sam wchodzi do głowy". Ci, którzy używają innych języków, często twierdzą, że brakuje im funkcjonalności, jakie oferuje Python. Tym różni się ten język od swoich konkurentów.

# Kiedy nie używać Pythona?

Python nie jest językiem dobrym na wszystko.

W większości sytuacji kod napisany w Pythonie działa wystarczająco szybko, ale zdarzają się szczególne przypadki, w których jest zbyt wolny. Jeżeli program, który zamierzasz napisać, ma głównie wykonywać obliczenia (używając fachowego żargonu, będzie *ograniczony przez procesor*), lepiej będzie, jeżeli użyjesz języka C, C++, C#, Java, Rust lub Go.

Nie jest to jednak regułą, ponieważ:

- Niektóre algorytmy stosowane w Pythonie są wydajniejsze niż na przykład w C. Ponadto dzięki temu, że programista w Pythonie tworzy kod szybciej niż w innych językach, ma on więcej czasu na eksperymentowanie z alternatywnymi rozwiązaniami.
- W wielu aplikacjach, szczególnie internetowych, kod bezczynnie oczekuje, aż serwer wyśle odpowiedź przez sieć. Procesor jest wtedy obciążony w znikomym stopniu i w efekcie różnica pomiędzy szybkością działania programów napisanych w językach statycznym i dynamicznym jest niewielka.
- Interpreter Pythona jest napisany w języku C i można go rozszerzać o ten kod. Ten temat jest poruszony w rozdziale 19.
- Interpreter Pythona jest coraz szybszy. Język Java w swoim dziecięcym wieku był nieznośnie powolny i w jego przyspieszenie włożono mnóstwo wysiłku i pieniędzy. Python nie jest własnością żadnej firmy, więc usprawnienia pojawiają się stopniowo. W rozdziale 19. opisany jest projekt PyPy i uzyskane dzięki niemu efekty.

Czasami jednak wymagania stawiane przed aplikacją są tak wysokie, że za pomocą Pythona nie można ich spełnić, nawet bardzo się starając. Zazwyczaj wyjściem jest wtedy użycie C, C++ lub Javy. Warto również przyjrzeć się językom Go (który jest podobny do Pythona i wydajny jak C) i Rust.

# Wersje Pythona 2 i 3

Pewną komplikację stanowi to, że są dwie wersje Pythona. Wersja 2 istnieje od zawsze i jest standardowo instalowana w systemach Linux i macOS. Jest świetna, ale nie idealna. W informatyce, tak jak w wielu innych dziedzinach, niektóre kosmetyczne błędy łatwo jest poprawić, inne trudniej. Wprowadzanie dużych poprawek powoduje, że kolejne wersje przestają być *kompatybilne*. Programy napisane przy użyciu nowszej wersji mogą nie działać w starszych środowiskach i odwrotnie.

Twórca Pythona, Guido van Rossum (*https://www.python.org/~guido*), wraz ze współpracownikami postanowił zebrać wszystkie poprawki i w 2008 r. udostępnił nową wersję języka Python 3. Od tamtej pory Python 2 jest uważany za przeszłość, a Python 3 za przyszłość. Najnowsza podwersja Pythona 2, oznaczona numerem 2.7, będzie jeszcze istnieć przez jakiś czas, ale jest ostatnia. Podwersji 2.8 już nie będzie. Utrzymywanie Pythona 2 i wielu ważnych pakietów zakończyło się w styczniu 2020 r. i od tamtej pory nie są już wprowadzane poprawki (patrz *https://python3statement.org*). Wkrótce z systemów operacyjnych zostanie usunięta wersja 2 i domyślnie będzie instalowana wersja 3. Popularne programy są stopniowo konwertowane do nowej wersji i zdecydowana większość jest już do niej dostosowana. Wszystkie nowe projekty są teraz tworzone w wersji 3.

W niniejszej książce wykorzystana jest wersja 3, która wygląda tu niemal identycznie jak wersja 2. Jedna z najważniejszych różnic polega na tym, że w wersji 3 `print` jest funkcją i jej argumenty trzeba umieszczać w nawiasach. Oprócz tego inaczej są obsługiwane znaki Unicode, o czym będzie mowa w rozdziale 12. Inne różnice są opisywane na bieżąco w tekście.

# Instalacja Pythona

Python nie jest domyślnie zainstalowany w każdym systemie. Aby nie robić bałaganu w rozdziale, szczegółowe informacje, jak zainstalować Pythona, zostały umieszczone w dodatku B. Jeżeli używasz innej wersji Pythona niż 3 lub nie wiesz dokładnie jakiej, zajrzyj do tego dodatku i sprawdź, co musisz zrobić. Może to być dość skomplikowane, ale będziesz to musiał zrobić tylko raz.

# Uruchomienie Pythona

Po zainstalowaniu środowiska Python 3 możesz go użyć do uruchamiania programów opisanych w tej książce, jak również własnych. Jak się to robi? Poniżej przedstawione są dwa podstawowe sposoby.

- Z małymi programami można łatwo eksperymentować za pomocą wbudowanego **interaktywnego interpretera** (powłoki). Wpisuje się w nim kod wiersz po wierszu i natychmiast uzyskuje wyniki. Dzięki krótkiej drodze między wpisaniem kodu a uzyskaniem wyników można efektywniej eksperymentować. W tej książce używam interpretera do prezentowania funkcjonalności języka. Te same instrukcje możesz wpisywać we własnym środowisku.
- Większe programy zapisuje się w plikach tekstowych, zazwyczaj z rozszerzeniem *.py*, i uruchamia przez wpisanie polecenia `python` i nazwy pliku.

Wypróbujmy teraz obie metody.

## Interaktywny interpreter

W większości opisanych w tej książce przykładów wykorzystany jest interaktywny interpreter. Gdy będziesz wpisywał te same instrukcje jak w przykładach i uzyskiwał te same efekty, będzie to oznaczało, że idziesz właściwą drogą.

Interpreter uruchamia się przez wpisanie nazwy głównego pliku wykonywalnego Pythona: `python`, `python3` lub podobnej. W całej książce przyjęte jest założenie, że plik ten nazywa się `python`. Jeżeli w Twoim przypadku nazwa jest inna, wpisuj ją wszędzie tam, gdzie w przykładach jest użyta nazwa `python`.

Instrukcje wpisywane w interaktywnym interpreterze i w plikach działają niemal tak samo, z jednym wyjątkiem. Jeżeli w interpreterze wpiszesz instrukcję reprezentującą jakąś wartość, zostanie ona automatycznie wyświetlona. Nie jest to cecha Pythona, tylko interpretera, dzięki której nie trzeba za każdym razem wpisywać funkcji `print()`. Na przykład, gdy w interpreterze wpiszesz liczbę 27, pojawi się ona na ekranie. Jeżeli użyjesz jej w pliku, nic złego się nie stanie, ale liczba ta nigdzie się nie pojawi. Ilustruje to poniższy przykład.

```
$ python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 27
27
```

W tym przykładzie symbol $ oznacza *znak zachęty* do wprowadzenia polecenia (tutaj `python`). Jest on stosowany w przykładach opisanych w tej książce, jednak w Twoim systemie może być inny.

Aby wyświetlić coś w interpreterze, użyj funkcji `print()`, jak niżej:

```
>>> print(27)
27
```

Jeżeli wypróbowałeś opisane wcześniej przykłady w interaktywnym interpreterze i uzyskałeś takie same wyniki, to znaczy, że uruchomiłeś prawdziwe, choć małe, programy. W kolejnych rozdziałach będziesz stopniowo tworzył coraz dłuższe kody.

## Skrypty

Jeżeli liczbę 27 wpiszesz w pliku i uruchomisz go, wtedy na ekranie nic się nie pojawi. Jest to typowe działanie programów, które nie są interaktywne. Aby coś wyświetlić, musisz użyć funkcji `print()`, jak niżej:

```
print(27)
```

Napisz program w Pythonie i uruchom go w następujący sposób:

1. Otwórz edytor tekstowy.
2. Wpisz instrukcję `print(27)`, jak wyżej.

3. Zapisz plik pod nazwą z rozszerzeniem *.py*, na przykład *test.py*. Upewnij się, że zapisałeś go w zwykłym formacie tekstowym, a nie wzbogaconym, na przykład RTF lub Word. Stosowanie rozszerzenia *.py* nie jest obowiązkowe, jednak warto to robić, ponieważ dzięki niemu wiadomo, co to jest za plik.
4. Jeżeli korzystasz z graficznego interfejsu systemu operacyjnego — jak niemal wszyscy — otwórz terminal lub wiersz poleceń[2].
5. Uruchom program następującym poleceniem:

```
$ python test.py
```

Powinien pojawić się jeden wiersz z wynikiem:

```
27
```

Udało się? Jeżeli tak, to gratuluję Ci napisania i uruchomienia pierwszego programu w języku Python.

## Co dalej?

Teraz zaczniesz wpisywać polecenia języka Python zgodnie z obowiązującą w nim składnią. Nie zarzucę Cię od razu wszystkimi regułami, tylko będę je przedstawiał stopniowo w kolejnych rozdziałach.

Podstawowy sposób tworzenia programów w Pythonie polega na wpisywaniu kodu w zwykłym edytorze tekstu lub oknie terminala. W tej książce prezentowany jest zwykły tekst wpisywany w terminalu lub w pliku. Pamiętaj jednak, że dostępnych jest kilka niezłych środowisk IDE (ang. *Integrated Development Environment* — zintegrowane środowisko programistyczne), oferujących wygodę interfejsu graficznego i zaawansowanych technik edytowania i wyświetlania tekstu. Więcej na ten temat dowiesz się w rozdziale 19.

# Twoja chwila zen

Każdy język programowania ma własny styl. We wstępie wspomniałem, że istnieje pytoniczny sposób wyrażanie siebie. Interpreter Pythona zawiera w sobie kilkanaście sentencji oddających zwięźle filozofię tego języka (o ile mi wiadomo, Python jest jedynym językiem, który ma coś takiego). Aby przeżyć chwilę zen, wpisz w interaktywnym interpreterze polecenie `import this` i naciśnij klawisz *Enter*. Otrzymasz wynik podobny do poniższego:

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

[2] Jeżeli nie wiesz, co to jest, zajrzyj do dodatku B, w którym opisane są niuanse różnych systemów operacyjnych.

```
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Wszystkie przykłady prezentowane w tej książce nawiązują do powyższych zasad.

# Już wkrótce

Następny rozdział traktuje o typach danych i zmiennych. Stanowi przygotowanie do dalszych rozdziałów, w których zagłębimy się w szczegóły typów danych i struktury kodu.

# Do zrobienia

Niniejszy rozdział jest wprowadzeniem do języka Python — o tym, do czego służy, jak wygląda i jakie jest jego miejsce w informatycznym świecie. Na zakończenie proponuję Ci wykonanie kilku miniprojektów, dzięki którym utrwalisz dotąd zdobytą wiedzę i przygotujesz się do dalszych rozdziałów.

1.1. Jeżeli nie zainstalowałeś jeszcze Pythona 3, zrób to teraz. W dodatku B znajdziesz szczegółowe informacje dotyczące Twojego systemu.

1.2. Uruchom interaktywny interpreter Pythona 3. Jak poprzednio, szczegółowe informacje są dostępne w dodatku B. Interpreter wyświetli kilka wierszy informacji o sobie i znaki >>>, będące zachętą do wpisania instrukcji Pythona.

1.3. Poeksperymentuj trochę z interpreterem. Użyj go w charakterze kalkulatora. Wpisz na przykład **`8 * 9`** i naciśnij *Enter*. Powinien pojawić się wynik 72.

1.4. Wpisz liczbę **`47`** i naciśnij *Enter*. Czy w następnym wierszu pojawiła się ta sama liczba?

1.5. Wpisz instrukcję **`print(47)`** i naciśnij *Enter*. Czy tym razem też pojawiła się ta liczba?

# Problemy? Rozwiąż je po pythonowsku!

Python nie jest językiem idealnym, jednak przybywa programistów, którzy uważają go za bliski ideału. Wyróżnia się prostotą i wszechstronnością. Jest wdzięcznym narzędziem do badania danych i tworzenia systemów sztucznej inteligencji, uwielbiają go analitycy, ekonomiści i naukowcy. Może posłużyć do tworzenia stron WWW czy aplikacji specjalnego przeznaczenia. Python należy do najbardziej spójnych i czytelnych języków programowania. Jest przykładem całkiem udanego kompromisu pomiędzy prostotą, łatwością przyswajania i wyjątkową skutecznością. Z pewnością warto się go nauczyć, jednak od początku dobrze jest wpoić sobie nawyki pisania kodu nowoczesnego, wysokiej jakości, zgodnego z dobrą praktyką.

Oto znakomity, przystępny i świetnie napisany podręcznik do nauki Pythona. Opisuje podstawy kodu i struktur danych i stopniowo wprowadza bardziej zaawansowane zagadnienia, takie jak praca z bazami danych i stronami WWW, podstawy działania chmury obliczeniowej, uczenia maszynowego i strumieniowania zdarzeń. Poza standardową biblioteką Pythona przedstawiono tu przydatne zewnętrzne pakiety, dokładniej opisano te najbardziej pomocne. Omówiono dobre praktyki tworzenia, testowania i diagnozowania kodu. Książka zawiera też mnóstwo wskazówek i przykładów kodu. Wyjaśnia pewne szczególne funkcjonalności Pythona, których stosowanie jest o wiele lepszym rozwiązaniem niż adaptowanie technik z innych języków. Nawet jeśli dziś o programowaniu wiesz mniej niż niewiele, dzięki temu podręcznikowi staniesz się prawdziwym pythonowcem!

## W książce między innymi:

- podstawy Pythona oraz funkcje, moduły i pakiety
- programowanie zorientowane obiektowo
- praca z bazami danych: relacyjnymi i NoSQL
- klienty internetowe, serwery, interfejsy API i usługi
- zarządzanie programami, procesami i wątkami
- implementacja współbieżności i komunikacji sieciowej

**Bill Lubanovic** jest ekspertem w dziedzinie technologii informatycznych i programistą. Systemem Unix zajmuje się od 1977 roku, interfejsami graficznymi od 1981 roku, bazami danych od początku lat 90., a siecią WWW od 1993 roku. Mieszka z rodziną w górach Sangre de Sasquatch w Minnesocie w Stanach Zjednoczonych.


ISBN 978-83-283-6842-2