Python
w pigułce

Podręczny przewodnik
po wersjach 3.10 i 3.11

Alex Martelli,
Anna Martelli Ravenscroft,
Steve Holden i Paul McGuire

przekład: Marek Włodarz

APN Promise
Warszawa 2023

O'REILLY®

Python w pigułce. Podręczny przewodnik po wersjach 3.10 i 3.11

Authorized translation of English edition of
Python in the Nutshell
ISBN 978-1-098-11355-1

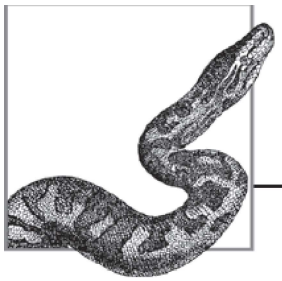
APN PROMISE SA, ul. Domaniewska 44a, 02-672 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: wydawnictwo@promise.pl

ISBN: 978-83-7541-528-5 (druk), 978-83-7541-529-2 (ebook)

Projekt okładki: Karen Montgomery
Ilustracje: Kate Dullea

Przekład: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Spis treści



Wprowadzenie

Język programowania Python łączy w sobie wiele pozornych sprzeczności: elegancki choć pragmatyczny, prosty choć skuteczny, bardzo wysokiego poziomu, ale nie sprzeciwi się, gdy zajdzie potrzeba podłubać w bitach bajtach, a do tego jest odpowiedni dla nowicjuszy i świetny dla ekspertów.

Książka ta jest przeznaczona dla programistów, którzy mieli już pewne doświadczenia z Pythonem, ale również dla doświadczonych programistów, którzy przechodzą do Pythona z innych języków. Została pomyślana jako podręczne kompendium samego Pythona, najczęściej używanych części ogromnej biblioteki standardowej oraz pewnej liczby najbardziej popularnych i użytecznych modułów i pakietów innego pochodzenia. Ekosystem Pythona rozrósł się już tak bardzo, jeśli chodzi o jego bogactwo, zakres i złożoność, że nie ma już nadziei, aby pojedynczy tom mógł być encyklopedycznym źródłem wiedzy. Nadal jednak książka obejmuje szeroki wybór obszarów zastosowań, w tym programowanie dla Internetu i sieci, obsługę XML, interakcje z bazami danych i obliczenia numeryczne. Eksploruje też możliwości międzyplatformowe oraz podstawy rozszerzania Pythona i osadzania jego kodu w innych aplikacjach.

Jak korzystać z tej książki

Choć można po prostu czytać ten tom liniowo od początku do końca, staraliśmy się też, aby był użytecznym źródłem informacji w pracy programisty. Do zlokalizowania interesujących tematów można użyć indeksu albo przeczytać wybrane rozdziały obejmujące konkretne tematy. Jakkolwiek Czytelnik będzie jej używał, szczerze życzymy przyjemnej lektury w czytaniu tego, co reprezentuje owoce przeszło rocznej pracy naszego zespołu.

Książkę można umownie podzielić na pięć części, jak poniżej.

Część I, Pierwsze kroki w Pythonie

Rozdział 1, „Wprowadzenie do języka Python”
: Przedstawia ogólną charakterystykę języka Python, jego implementacje, źródła pomocy i informacji technicznych, jak uczestniczyć w społeczności Pythona, oraz wskazówki, jak pobrać i zainstalować Pythona na swoim komputerze(ach) albo uruchamiać go w przeglądarce.

Rozdział 2, „Interpreter Pythona”
: Omawia działanie programu interpretera Pythona, jego opcje wiersza poleceń i jego użycie do uruchamiania programów w Pythonie i sesji interaktywnych. Rozdział wskazuje kilka edytorów tekstowych do pracy nad programami w Pythonie oraz programy pomocnicze do sprawdzania kodu źródłowego oraz kilka pełnokrwistych, zintegrowanych środowisk programistycznych, w tym IDLE, dostarczane bezpłatnie wraz ze standardową instalacją Pythona. Rozdział ten przedstawia też uruchamianie programów w Pythonie z wiersza poleceń.

Część II, Podstawowy język Python i elementy wbudowane

Rozdział 3, „Język Python”
: Przedstawia składnię Pythona, wbudowane typy danych, wyrażenia, instrukcje, sterowanie przepływem oraz pisanie i wywoływanie funkcji.

Rozdział 4, „Obiektowy Python”
: Omawia obiektowe podejście do programowania w Pythonie.

Rozdział 5, „Adnotacje typów”
: Wyjaśnia, jak dodawać informacje o typach do kodu Pythona, aby uzyskać wskazówki typów i autouzupełnianie w nowoczesnych edytorach kodu oraz wsparcie dla statycznego sprawdzania typów w programach sprawdzających i linterach.

Rozdział 6, „Wyjątki”
: Omawia wykorzystywanie wyjątków do obsługi błędów i sytuacji specjalnych, rejestrowania oraz tworzenie kodu, który automatycznie wykonuje sprzątanie w razie wystąpienia wyjątku.

Rozdział 7, „Moduły i pakiety”
: Pokazuje, jak grupować kod w moduły i pakiety, jak definiować i importować moduły i jak instalować pakiety Python innych firm/autorów. Rozdział ten omawia też pracę w środowisku wirtualnym w celu wyizolowania zależności projektu.

Rozdział 8, „Obiekty wbudowane i moduły biblioteki standardowej”
: Przedstawia wbudowane typy danych oraz funkcje oraz część z najbardziej fundamentalnych modułów biblioteki standardowej Pythona (mówiąc w uproszczeniu,

chodzi o zbiór modułów zapewniający funkcjonalność, która w niektórych innych językach jest wbudowana w sam język).

Rozdział 9, „Ciągi i rzeczy”
: Omawia narzędzia Pythona do przetwarzania ciągów, w tym ciągów Unicode, ciągów bajtowych oraz literałów tekstowych.

Rozdział 10, „Wyrażenia regularne”
: Przedstawia obsługę wyrażeń regularnych w Pythonie.

Część III, Biblioteka Pythona i moduły rozszerzeń

Rozdział 11, „Operacje na plikach i tekście”
: Zawiera omówienie przetwarzania plików i tekstu za pomocą wielu modułów biblioteki standardowej Pythona oraz specyficznych dla platform rozszerzeń dla tekstu formatowanego. Rozdział przedstawia też problemy związane z lokalizacjami i internacjonalizacją.

Rozdział 12, „Utrwalanie i bazy danych”
: Przedstawia mechanizmy serializacji i utrwalania w Pythonie oraz interfejsy dla baz danych DBM i relacyjnych (opartych na SQL) baz danych, a w szczególności dla poręcznego SQLite dostarczanego waz z biblioteką standardową Pythona.

Rozdział 13, „Operacje daty i czasu”
: Pokazuje, jak sobie radzić z danymi daty i czasu przy użyciu biblioteki standardowej i rozszerzeń innych firm.

Rozdział 14, „Dostosowywanie wykonywania”
: Omawia sposoby osiągnięcia zaawansowanej kontroli wykonywania kodu w Pythonie, w tym wykonywanie dynamicznie generowanego kodu oraz sterowanie sprzątaniem pamięci. Rozdział ten omawia też niektóre wewnętrzne typy Pythona i konkretny problem rejestrowania funkcji „sprzątających” do wykonania podczas kończenia działania programu.

Rozdział 15, „Współbieżność: wątki i procesy”
: Przedstawia funkcjonalność Pythona dla wykonywania współbieżnego, zarówno poprzez wiele wątków działających w obrębie jednego procesu, jak i wiele procesów działających na jednej maszynie[1]. Rozdział ten pokazuje też, jak uzyskiwać dostęp do środowiska procesu i jak odwoływać się do plików poprzez mechanizmy mapowania pamięci.

1 Oddzielny rozdział na temat programowania asynchronicznego z wydania trzeciego został pominięty w tym wydaniu, a bardziej pogłębione omówienie tego rozrastającego się tematu zostało odłożone do źródeł wymienionych w rozdziale 15.

Rozdział 16, „Przetwarzanie numeryczne”

Przedstawia funkcjonalności Pythona używane w obliczeniach numerycznych, zarówno w modułach biblioteki standardowej, jak i pakietach rozszerzeń innych firm; w szczególności pokazujemy tu, jak używać liczb dziesiętnych lub ułamków zamiast domyślnych binarnych liczb zmiennoprzecinkowych. Rozdział pokazuje też, jak uzyskiwać i używać liczb pseudolosowych i prawdziwie losowych oraz jak szybko przetwarzać całe tablice (i macierze) liczb.

Rozdział 17, „Testowanie, debugowanie i optymalizowanie”

Omawia narzędzia i podejścia, które pomagają upewnić się, że nasze programy są poprawne(czyli że robią to, co miały robić), wyszukiwać i poprawiać błędy w programach oraz sprawdzać i ulepszać wydajność programów. Rozdział przedstawia też koncepcję ostrzeżeń oraz moduł biblioteki Pythona, który je obsługuje.

Część IV, Programowanie Internetu i sieci

Rozdział 18, „Podstawy sieci”

Omawia podstawy funkcji sieciowych w Pythonie.

Rozdział 19, „Klienckie moduły protokołów sieciowych”

Przedstawia moduły biblioteki standardowej Pythona do pisania sieciowych programów klienckich, w szczególności do obsługi różnych protokołów sieciowych po stronie klienta, wysyłanie i odbieranie emaili i obsługę adresów URL.

Rozdział 20, „Serwowanie HTTP”

Pokazuje, jak udostępniać HTTP na potrzeby aplikacji webowych, używając popularnych lekki frameworków wykorzystujących standardowy interfejs WSGI Pythona dla serwerów webowych.

Rozdział 21, „Email, MIME i inne kodowania sieciowe”

Wyjaśnia, jak w Pythonie przetwarzać wiadomości email i inne strukturyzowane sieciowo i kodowane dokumenty.

Rozdział 22, „Tekst strukturalny: HTML”

Omawia popularne moduły rozszerzeń Pythona innych firm do przetwarzania, modyfikowania i generowania dokumentów HTML.

Rozdział 23, „Tekst strukturalny: XML”

Omawia moduły biblioteki Pythona i popularne rozszerzenia dla przetwarzania, modyfikowania i generowania dokumentów XML.

Część V, Rozszerzanie, dystrybucja oraz aktualizacja wersji i migracja

Rozdziały 24 i 25 w drukowanym wydaniu tej książki zostały zamieszczone w formie podsumowań. Pełną treść tych rozdziałów można znaleźć w powiązanym z książką repozytorium opisanym w podrozdziale „Jak się z nami kontaktować” na stronie xv.

Rozdział 24, „Pakowanie programów i rozszerzeń”
: Omawia narzędzia i moduły do tworzenia pakietów i udostępniania modułów i aplikacji Pythona.

Rozdział 25, „Rozszerzanie i osadzanie Classic Python”
: Omawia, jak programować moduły rozszerzeń Pythona przy użyciu C API, Cythona i innych narzędzi.

Rozdział 26, „Migracja z wersji 3.7 do 3.n”
: Omawia zagadnienia i najlepsze praktyki planowania i wdrażania aktualizacji wersji dla użytkowników Pythona od indywidualnych osób, poprzez opiekunów bibliotek, po wdrożenia na poziomie przedsiębiorstwa i zespołów wsparcia.

Dodatek A, „Nowe funkcjonalności i zmiany w Pythonie od wersji 3.7 do 3.11”
: Udostępnia szczegółową listę zmian funkcjonalności i składni języka Python i biblioteki standardowej dla poszczególnych wersji.

Konwencje użyte w tej książce

W tej książce używane są rozmaite konwencjonalne zapisy.

Konwencje odwołań

Gdy jest to wykonalne, we wpisach odwołań do funkcji/metod każdy parametr opcjonalny jest pokazywany z wartością domyślną przy użyciu składni Pythona *`name=value`*. Wbudowane funkcje nie muszą akceptować nazwanych parametrów, zatem nazwy tych parametrów nie muszą być znaczące. Niektóre opcjonalne parametry najlepiej wyjaśnia ich obecność lub brak, a nie ich domyślne wartości. W takich przypadkach zaznaczamy opcjonalność parametru, umieszczając go w nawiasach kwadratowych (`[]`). Nawiasy te mogą być zagnieżdżane, jeśli więcej niż jeden argument jest opcjonalny.

Konwencje wersji

Książka omawia zmiany i funkcjonalności Pythona w wersjach od 3.7 do 3.11. Python 3.7 służy jako wersja bazowa dla wszystkich tabel i przykładów kodu, o ile nie zostanie to oznaczone oddzielnie[2]. W treści książki można zauważyć poniższe oznaczenia wskazujące zmiany albo funkcjonalności dodane lub usunięte w poszczególnych z omawianych wersji:

- **`3.x+`** oznacza funkcjonalność wprowadzoną w wersji 3.x, niedostępną w wersjach wcześniejszych.

2 Dla przykładu, aby dostosować się do rozległych zmian, które nastąpiły w Pythonie 3.9 i 3.10 w adnotacjach typów, większa część rozdziału 5 używa Pythona 3.10 jako wersji bazowej przy omawianiu funkcjonalności i w przykładach.

- **-3.x** oznacza funkcjonalność usuniętą w wersji 3.x, dostępną tylko w wersjach wcześniejszych.

Konwencje typograficzne

Proszę zwrócić uwagę, że ze względu na ograniczenia drukowanej strony nasze fragmenty kodu i przykłady mogą niekiedy odchodzić od zaleceń PEP 8 (*https://oreil.ly/u0RLL*). Nie zalecamy pozwalania sobie na taką swobodę w rzeczywistym kodzie. Zamiast tego warto użyć narzędzi, takich jak `black` (*https://oreil.ly/BM68x*), aby wprowadzić kanoniczny styl układu.

W tej książce używane są następujące konwencje typograficzne:

Kursywa
: Używana dla nazw plików, katalogów, programów, adresów URL, a także przy wprowadzaniu nowych terminów i pojęć.

`Czcionka stałopozycyjna`
: Służy do prezentacji wpisywanych poleceń, przykładów kodu, a także do wyróżnienia elementów kodu występujących w tekście, takich jak nazwy metod, funkcji, klas i modułów.

`Pochylona czcionka stałopozycyjna`
: Używana w przykładach kodu i poleceń dla tekstu, który ma być zastąpiony wartościami dostarczonymi przez użytkownika.

`Wytłuszczona czcionka stałopozycyjna`
: Używana dla poleceń wpisywanych w wierszu poleceń oraz dla danych wyjściowych w przykładach sesji interpretera Pythona. Dodatkowo używana jest dla słów kluczowych Pythona.

Ten element oznacza wskazówkę lub sugestię.

Ten element oznacza uwagę ogólną.

Ten element wskazuje ostrzeżenie lub przestrogę.

Korzystanie z przykładów kodu

Celem tej książki jest wsparcie wykonywanej pracy. Ogólnie rzecz ujmując, przykładowe kody dołączone do książki można używać w programach i dokumentacji. Nie trzeba się kontaktować z wydawnictwem w celu uzyskania pozwolenia, chyba że jest wykorzystywana znaczna część kodu. Na przykład napisanie programu, który używa kilka fragmentów kodu z tej książki, nie wymaga pozwolenia. Sprzedaż lub dystrybucja przykładów z książek wydawnictwa O'Reilly wymaga już pozwolenia. Odpowiedź na pytanie poprzez zacytowanie tej książki i wykorzystanie przykładowego kodu nie wymaga pozwolenia. Włączenie znacznej ilości przykładowego kodu z tej książki do dokumentacji produktu wymaga pozwolenia.

Doceniamy uznanie autorstwa, ale nie wymagamy go. Zazwyczaj obejmuje ono tytuł, autora, wydawcę i numer ISBN. Na przykład: „*Python in a Nutshell*, 4th ed., by Alex Martelli, Anna Martelli Ravenscroft, Steve Holden, and Paul McGuire. Copyright 2023, 978-1-098-11355-1".

W przypadku, gdy wykorzystanie przykładów kodu może wykraczać poza dozwolony użytek lub powyżej podane zezwolenie, należy się skontaktować z wydawnictwem poprzez wysłanie maila na adres *permissions@oreilly.com*.

Jak się z nami kontaktować

Książka ma swoje własne repozytorium na GitHubie, w którym zamieszczana jest errata, przykłady oraz dowolne informacje dodatkowe. Repozytorium to zawiera też pełną treść rozdziałów 24 i 25, dla których zabrakło miejsca w drukowanym tomie. Repozytorium to jest dostępne pod adresem *https://github.com/pynutshell/pynut4*.

Dołożyliśmy wszelkich starań, aby jak najdokładniej przetestować i zweryfikować informacje zawarte w tej książce, ale Czytelnik może odkryć, że pewne funkcje się zmieniły (albo że popełniliśmy pomyłki!). Prosimy o poinformowanie wydawcy o dowolnych znalezionych błędach, a także o przekazywanie sugestii dotyczących przyszłych wydań.

Wydawnictwo O'Reilly utrzymuje stronę web dla tej książki, w której znajduje się errata, przykłady i informacje dodatkowe. Strona ta jest dostępna pod adresem *https://oreil.ly/python-nutshell-4e*.

Komentarze i pytania techniczne dotyczące książki można wysyłać na adres: *pynut4@gmail.com*.

Więcej informacji o naszych książkach, kursach, konferencjach i wiadomościach, zobacz *https://oreilly.com*.

Znajdź nas na LinkedIn: *https://linkedin.com/company/oreilly-media*.

Śledź nas na Twitterze: *https://twitter.com/oreillymedia*.

Oglądaj nas na YouTube: *https://www.youtube.com/oreillymedia*.

Podziękowania

Chcielibyśmy podziękować redaktorom i pracownikom O'Reilly: Amandzie Quinn, Brianowi Guerin, Zanowi McQuade i Kristen Brown. Szczególne podziękowania należą się naszej redaktorce, Angeli Rufino, która wykonała wielką pracę, aby zapewnić realizację tej książki w ustalonym czasie! Dziękujemy też wspaniałej korektorce Rachel Head za to, że wydajemy się bardziej wyedukowani, niż jesteśmy, oraz redaktorowi technicznemu Christopherowi Faucherowi za sprawienie, że książka wygląda tak dobrze zarówno w drukowanej, jak i elektronicznej postaci.

Dziękujemy naszym ciężko pracującym recenzentom technicznym, Davidowi Mertzowi, Markowi Summerfieldowi i Pankaj Gaijar, którzy dokładnie sczytali każde objaśnienie i przykład w rękopisie książki. Bez ich pomocy książka nie byłaby tak dokładna[3]. Za wszystkie błędy, które zostały, ponosimy wyłączną odpowiedzialność.

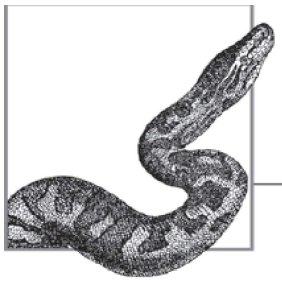
Dziękujemy też Luciano Ramalho, całemu zespołowi PyPy, Sebastiánowi Ramírezowi, Fabio Pligerowi, Miguelowi Grinbergowi oraz zespołowi Python Packaging Authority za ich pomoc w wybranych częściach książki, a także Google za ich użyteczne narzędzia współpracy online, bez których nasza intensywna łączność (i koordynacja pracy autorów przebywających na różnych kontynentach!) byłaby znacznie mniej przyjemna i wydajna.

Na koniec autorzy i wszyscy czytelnicy tej książki winni są wdzięczność kluczowym deweloperom samego języka Python – bez ich heroicznej pracy nie byłoby potrzeby ani możliwości jej napisania.

3 Ani nie zawierałaby tak wielu przypisów!

Część I

Pierwsze kroki w Pythonie



1

Wprowadzenie do języka Python

Python jest dobrze ugruntowanym językiem programowania ogólnego przeznaczenia, opublikowanym po raz pierwszy w roku 1991 przez swojego twórcę, Guido van Rossuma. Ten stabilny i dojrzały język jest językiem wysokiego poziomu, dynamicznym, obiektowym i międzyplatformowym – same bardzo atrakcyjne cechy. Python działa w systemie macOS, w większości dzisiejszych wariantów Unixa, w tym w Linuksie, w Windows, a po pewnym dostrojeniu na platformach mobilnych[1].

Python oferuje wysoką produktywność we wszystkich fazach cyklu życia oprogramowania: analizie, projektowaniu, prototypowaniu, kodowaniu, testowaniu, debugowaniu, dostrajaniu, dokumentowaniu i, oczywiście, utrzymywaniu. Język w ciągu lat jest coraz bardziej popularny, stając się liderem indeksu TIOBE (*https://oreil.ly/qxdeK*) w październiku 2021. Obecnie znajomość Pythona jest plusem dla każdego programisty: zakradł się już do większości nisz i obszarów zastosowań, odkrywając użyteczne role w dowolnych rozwiązaniach programistycznych.

Python zapewnia unikatowy miks elegancji, prostoty, praktyczności i czystej mocy. Można bardzo szybko stać się skutecznym programistą Pythona dzięki jego spójności i regularności, bogatej bibliotece standardowej oraz wielu pakietom i narzędziom innych firm, które są natychmiast dostępne. Python jest łatwy w nauce, zatem nadaje się doskonale dla tych, którzy dopiero zaczynają programowanie, a zarazem dość potężny, aby zaspokoić najbardziej wyrafinowanych ekspertów.

1 W przypadku Androida patrz *https://wiki.python.org/moin/Android*, zaś dla iPhone i iPada patrz *Python for iOS and iPadOS* (*https://oreil.ly/iYnk3*).

Język Python

Język Python, choć nie minimalistyczny, jest oszczędny z dobrych, pragmatycznych powodów. Gdy język oferuje jeden dobry sposób, aby wyrazić potrzeby projektu, dodawanie innych sposobów zapewnia w najlepszym razie umiarkowane korzyści. Natomiast koszt złożoności języka rośnie szybciej niż liniowo wraz z liczbą funkcjonalności. Skomplikowany język jest trudniejszy do nauczenia i opanowania (a także do skutecznego implementowania bez usterek), niż prosty. Złożoności i sztuczki w języku krępują produktywność podczas tworzenia oprogramowania, szczególnie w dużych projektach, przy których wielu deweloperów pracuje wspólnie i często utrzymuje kod oryginalnie napisany przez kogoś innego.

Python jest dość prosty, ale nie uproszczony. Jest to zgodne z ideą, że jeśli język zachowuje się w pewien sposób w pewnych kontekstach, w idealnym wariancie powinien działać analogicznie we wszystkich kontekstach. Python przestrzega zasady, że język nie powinien mieć „wygodnych" skrótów, specjalnych przypadków, wyjątków ad hoc, nazbyt subtelnych rozróżnień ani tajemniczych, skrytych optymalizacji działających w tle. Dobry język, tak jak każdy dobrze zaprojektowany artefakt, musi wyważać ogólne reguły ze smakiem, zdrowym rozsądkiem i mnóstwem praktyczności.

Python jest językiem programowania ogólnego przeznaczenia: jego cechy są użyteczne w niemal każdej dziedzinie wytwarzania oprogramowania. Nie istnieje taki obszar, w którym Python nie mógłby być częścią rozwiązania. „Część" jest tu ważna; choć wielu deweloperów uznaje, że Python wypełnia ich wszystkie potrzeby, nie musi zawsze stać samodzielnie. Programy w Pythonie mogą współpracować z wieloma innymi komponentami programistycznymi, co czyni go odpowiednim językiem do sklejania ze sobą fragmentów napisanych w innych językach. Jednym z celów projektowych języka od dawna jest „dobrze współgrać z innymi".

Python jest językiem bardzo wysokiego poziomu (*very high-level language* – VHLL). Oznacza to, że używa wyższego poziomu abstrakcji, koncepcyjnie bardziej oddalonego od maszyny działającej w tle, niż klasyczne kompilowane języki, takie jak C, C++ i Rust, tradycyjnie nazywane „językami wysokiego poziomu". Python jest prostszy, szybszy do przetworzenia (zarówno dla ludzi, jak i narzędzi) i bardziej regularny, niż klasyczne języki wysokiego poziomu. Zapewnia to wysoką produktywność programisty, co czyni Pythona potężnym narzędziem deweloperskim. Dobre kompilatory dla klasycznych języków kompilowanych mogą generować kod binarny, który działa szybciej niż Python. Jednak w większości przypadków wydajność aplikacji zakodowanych w Pythonie jest wystarczająca. Kiedy tak nie jest, można zastosować techniki optymalizacyjne omówione w punkcie „Optymalizacja" na stronie 609, aby poprawić wydajność swojego programu, jednocześnie zachowując korzyści wysokiej produktywności.

Pod względem poziomu abstrakcji, Python jest porównywalny z innymi efektywnymi językami VHLL, takimi jak JavaScript, Ruby i Perl. Jednak zalety prostoty i regularności pozostają po stronie Pythona.

Python jest obiektowym językiem programowania (*object-oriented* –OO), ale pozwala programować zarówno w stylu obiektowym, jak i proceduralnym, z pewną dozą programowania funkcyjnego, mieszając i dopasowując te style zgodnie z wymaganiami aplikacji. Cechy obiektowe Pythona są konceptualnie podobne do tych z C++, ale prostsze w użyciu.

Standardowa biblioteka Pythona i moduły rozszerzeń

Programowanie w Pythonie to coś więcej, niż tylko język: biblioteka standardowa i inne moduły rozszerzeń są niemal równie ważne w posługiwaniu się Pythonem, jak sam język. Biblioteka standardowa Pythona dostarcza wielu dobrze zaprojektowanych, solidnych modułów Pythona do wygodnego wykorzystania. Zawiera moduły dla takich zadań, jak prezentacja danych, przetwarzanie tekstu, interakcja z systemem operacyjnym i systemem plików lub programowanie web, które działają na wszystkich platformach obsługiwanych przez Python.

Moduły rozszerzeń, czy to pochodzące z biblioteki standardowej, czy z innych źródeł, pozwalają kodowi Pythona na dostęp do funkcjonalności dostarczanych przez działający w tle system operacyjny lub inne komponenty programowe, takie jak graficzne interfejsy użytkownika (GUI), bazy danych lub sieci. Rozszerzenia zapewniają również wielką prędkość wykonywania zadań intensywnych obliczeniowo, takich jak parsowanie XML i przetwarzanie tablic numerycznych. Moduły rozszerzeń, które nie są napisane w Pythonie, niekoniecznie jednak korzystają z tej samej przenośności między platformami, jak czysty kod Pythona.

Moduły rozszerzeń można pisać w językach niższego poziomu, aby zoptymalizować wydajność niewielkich, ale intensywnych obliczeniowo części programu, które oryginalnie były prototypowane w Pythonie. Można też użyć takich narzędzi, jak Cython, ctypes i CFFI do opakowania istniejących bibliotek C/C++ w moduły rozszerzeń Pythona, co zostało omówione w punkcie „Extending Python Without Python's C API" w rozdziale 25 (dostępnym online – *https://oreil.ly/python-nutshell-25*). Można również osadzić kod Pythona w aplikacjach napisanych w innych językach, eksponując funkcjonalność aplikacji kodowi Pythona za pośrednictwem specyficznych dla aplikacji modułów rozszerzeń.

Książka ta dokumentuje wiele modułów z biblioteki standardowej i innych źródeł dla programowania sieciowego po stronie klienta i serwera, łączności z bazami danych, przetwarzania tekstu i plików binarnych oraz interakcji z systemami operacyjnymi.

Implementacje Pythona

W chwili pisania tych słów Python dysponuje dwiema pełnymi implementacjami o jakości produkcyjnej (CPython oraz PyPy), a także kilkoma nowszymi o wysokiej wydajności, które są jeszcze we wcześniejszych fazach wytwarzania, takimi jak Nuitka (*https://nuitka.net*), RustPython (*https://oreil.ly/1oUWk*), GraalVM Python (*https://oreil.ly/1XRt*) i Pyston (*https://www.pyston.org*), których jednak nie będziemy bliżej omawiać. W punkcie „Inne

opracowania, implementacje i dystrybucje" na stronie 7 wymieniamy również kilka innych implementacji w jeszcze wcześniejszych fazach rozwoju.

Książka ta głównie dotyczy CPython, najszerzej używanej implementacji, którą zazwyczaj dla uproszczenia nazywamy po prostu „Python". Jednak rozróżnienie pomiędzy językiem a jego implementacją jest ważne!

CPython

Classic Python (*https://www.python.org*) – znany też jako CPython, a często nazywany po prostu Python – to najbardziej aktualna, solidna i kompletna implementacja o jakości produkcyjnej języka Python. Jest to „implementacja referencyjna" języka. CPython jest kompilatorem kodu bajtowego, interpreterem oraz zbiorem wbudowanych i opcjonalnych modułów, a wszystko to zostało zakodowane w standardowym języku C.

CPython może być używany na każdej platformie, na której dostępny jest kompilator C zgodny ze standardem ISO/IEC 9899:1990[2] (co oznacza wszystkie nowoczesne, popularne platformy). W punkcie „Instalacja" na stronie 19 wyjaśniamy, jak pobrać i zainstalować CPython. Cała treść tej książki z wyjątkiem kilku wyraźnie oznaczonych podrozdziałów ma zastosowanie do CPython. W chwili pisania tych słów aktualna, właśnie opublikowana wersja CPython to 3.11.

PyPy

PyPy (*https://pypy.org*) to szybka i elastyczna implementacja Pythona, zakodowana przy użyciu podzbioru samego Pythona, zdolna do odwoływania się do wielu języków niższego poziomu oraz maszyn wirtualnych przy użyciu zaawansowanych technik, takich jak wnioskowanie typów. Największą siłą PyPy jest jego zdolność do generowania natywnego kodu maszynowego „na poczekaniu" (*just in time*, JIT), podczas uruchamiania programu w Pythonie; zapewnia to znaczące korzyści pod względem szybkości wykonywania. PyPy aktualnie implementuje wersję języka 3.8 (wersja 3.9 jest w fazie beta).

Wybieranie pomiędzy CPython, PyPy i innymi implementacjami

Jeśli nasza platforma, tak jak większość, jest w stanie uruchamiać CPython, PyPy i wiele innych implementacji Pythona, o których tu wspominamy, jak dokonać wyboru pomiędzy nimi? Nade wszystko, nie należy wybierać pochopnie: można pobrać i zainstalować je wszystkie. Mogą koegzystować ze sobą bez żadnych problemów i wszystkie są darmowe (choć niektóre oferują również wersje komercyjne z dodaną wartością, taką jak wsparcie techniczne, ale odpowiadające im wersje bezpłatne również są dobre). Utrzymywanie ich wszystkich na komputerze programisty kosztuje tylko nieco czasu pobierania i trochę przestrzeni dyskowej, a umożliwia bezpośrednie porównania. Co powiedziawszy, warto zwrócić uwagę na kilka ogólnych wskazówek.

2 Wersje Python od 3.11 używają „C11 bez opcjonalnych funkcjonalności" i określają, że „publiczne API powinno być kompatybilne z C++".

Jeśli potrzebujemy niestandardowej wersji Pythona albo wysokiej wydajności w długo działających programach, warto rozważyć PyPy (albo jeśli możemy się zgodzić na wersje, które jeszcze nie są w pełni gotowe do zastosowań produkcyjnych, jedną z innych wymienionych).

W przypadku pracy głównie w tradycyjnym środowisku CPython jest doskonałym dopasowaniem. Jeśli nie mamy silnych preferencji alternatywnych, należy zacząć od standardowej referencyjnej implementacji CPython, która jest najszerzej wspierana przez dodatki i rozszerzenia innych firm oraz oferuje najbardziej aktualną wersję.

Innymi słowy, aby eksperymentować, uczyć się i próbować różnych rzeczy, użyjemy CPython. W przypadku tworzenia i wdrażania najlepszy wybór zależy od modułów rozszerzeń, których chcemy użyć i tego, jak chcemy dystrybuować swoje programy. CPython z definicji wspiera wszystkie rozszerzenia Pythona; jednak PyPy wspiera większość z nich, a często może być szybszy w przypadku długo działających programów dzięki kompilacji w locie do kodu maszynowego – aby to sprawdzić, można porównać sprawność kodu w CPython z tą uzyskiwaną w PyPy (a dla pewności, także w innych implementacjach).

CPython jest najbardziej dojrzały: jest obecny od dłuższego czasu, podczas gdy PyPy i inne implementacje są nowsze, a tym samym mniej sprawdzone w praktyce. Opracowywanie wersji CPython następuje przed wdrożeniem ich dla innych implementacji.

PyPy, CPython i inne wspominane tu implementacje wszystkie są dobrymi, sumiennymi implementacjami języka Python, rozsądnie bliskie sobie nawzajem pod względem użyteczności i wydajności. Rozsądne jest poznanie zalet i słabości każdej z nich, a następnie dokonanie optymalnego wyboru dla każdego zadania programistycznego.

Inne opracowania, implementacje i dystrybucje

Python stał się tak popularny, że wiele grup i osób zainteresowało się jego rozwojem i dostarcza funkcjonalności oraz implementacji wykraczających poza to, na czym skupia się główny zespół deweloperski.

Obecnie większość systemów bazujących na Unixie zawiera jakąś implementację Pythona – typowo wersję 3.*x* dla pewnej wartości *x* – jako „systemowy Python". Aby mieć Pythona w Windows lub macOS, zazwyczaj trzeba pobrać i uruchomić instalator (*https://oreil.ly/c-TxU*) (zobacz także notatkę „macOS" na stronie 20). Jeśli ktoś poważnie myśli o tworzeniu oprogramowania w Pythonie, pierwszą rzeczą, którą trzeba zrobić, jest *pozostawienie w spokoju Pythona zainstalowanego przez system!* Poza wszystkim innym, Python jest coraz częściej i intensywniej używany przez pewne części samego systemu operacyjnego, zatem dłubanie w tej domyślnej instalacji Pythona może sprowadzić kłopoty.

Tym samym, nawet jeśli nasz system jest wyposażony w „systemowego Pythona", należy rozważyć zainstalowanie jednej lub więcej implementacji Pythona, aby móc ich swobodnie używać dla wygody programowania, bezpieczni dzięki wiedzy, że nic, co w nich zrobimy, nie wpłynie na działanie systemu operacyjnego. Gorąco polecamy też używanie *wirtualnych środowisk* (zobacz „Środowiska Pythona" na stronie 267) w celu odizolowania

projektów od siebie i pozwolić im na dysponowanie czymś, co inaczej tworzyłoby konfliktowe zależności (czyli sytuacje, w której dwa projekty wymagają różnych wersji tego samego modułu innej firmy). Alternatywnie możliwe jest zainstalowanie lokalnie wielu Pythonów obok siebie.

Popularność Pythona doprowadziła też do powstania wielu społeczności i ekosystem języka jest bardzo aktywny. W kolejnych podrozdziałach przedstawimy niektóre z bardziej interesujących osiągnięć: zwróćmy uwagę, że fakt pominięcia jakiegoś projektu w tym miejscu odzwierciedla raczej nasze ograniczenia co do miejsca (w książce) i czasu (na jej pisanie), a nie oznacza dezaprobaty!

Jython i IronPython

Jython (*https://www.jython.org*), obsługujący język Python na podstawie JVM (*https://oreil.ly/Q8EQB*) oraz IronPython (*https://ironpython.net*), który obsługuje Pythona w środowisku .NET (*https://oreil.ly/o_MTn*), to projekty open source, które, choć oferują jakość produkcyjną dla wspieranych przez nie wersji języka Python, wydawały się być „wstrzymane" w czasie pisania tych słów, jako że ostatnie obsługiwane przez nie wersje są znacząco opóźnione w stosunku do CPython. Każdy „wstrzymany" projekt open source może potencjalnie powrócić do życia: wszystko, czego potrzeba, to jeden lub więcej entuzjastycznych, zaangażowanych deweloperów, którzy poświęcą swój czas i zapał na jego „ożywienie". Jako alternatywę dla Jythona na JVM można również rozważyć wspomnianą wcześniej implementację GraalVM Python.

Numba

Numba (*https://numba.pydata.org*) to kompilator just-in-time (JIT) open source, który tłumaczy pewien podzbiór Pythona i NumPy. Biorąc pod uwagę, że skupia się głównie na przetwarzaniu numerycznym, powrócimy do niego ponownie w rozdziale 16.

Pyjion

Pyjion (*https://oreil.ly/P7wKC*) to projekt open source, oryginalnie rozpoczęty przez Microsoft z kluczowym celem dodania do CPython API do zarządzania kompilatorami JIT. Cele pomocnicze obejmują zaoferowanie kompilatora JIT dla środowiska CLR Microsoftu (*https://oreil.ly/5zjOG*) (które jest częścią .NET) i frameworka do opracowywania kompilatorów JIT. Pyjion nie ma na celu *zastąpienia* CPython; jest to raczej moduł importowany z CPython (aktualnie wymaga wersji 3.10), który pozwala przetłumaczyć kod bajtowy CPython „w locie" na kod maszynowy dla wielu różnych środowisk. Integracja Pyjion z CPython została umożliwiona przez PEP 523 (*https://oreil.ly/lFDGw*); jednak ponieważ zbudowanie Pyjiona wymaga wielu narzędzi oprócz kompilatora C (który jest wszystkim, czego potrzeba do zbudowania CPython), Python Software Foundation (PSF) zapewne nigdy nie włączy Pyjiona do dystrybuowanych przez siebie wydań CPython.

IPython

IPython (*https://ipython.org*) ulepsza interaktywny interpreter CPython, aby był wygodniejszy i bardziej skuteczny. Dopuszcza skróconą składnię wywołania funkcji oraz rozszerzalną funkcjonalność znaną jako *magia* (*magics*) wprowadzaną przy użyciu znaku procenta (`%`). Udostępnia też znaki ucieczki dla powłoki, co pozwala zmiennej Pythona odebrać wynik polecenia powłoki systemowej. Do odpytania dokumentacji obiektu należy użyć znaku zapytania (albo dwóch znaków zapytania dla rozszerzonej dokumentacji); naturalnie nadal dostępne są wszystkie standardowe funkcjonalności interaktywnego interpretera Pythona.

IPython poczynił szczególne postępy w świecie naukowym i skupionym na danych. Poprzez wynalezienie IPython Notebook, obecnie po refaktoryzacji przemianowanego na Jupyter Notebook, omówionego w punkcie „Jupyter" na stronie 37 stopniowo przekształcił się w interaktywne środowisko programistyczne, które oprócz fragmentów kodu[3] pozwala również osadzać komentarze w stylu *literate programming* (*https://oreil.ly/tx5B3*). Komentarze takie mogą zawierać dowolny tekst, włącznie z notacją matematyczną i pokazywać wyniki wykonania kodu, opcjonalnie nawet z grafiką generowaną przez takie podsystemy, jak `matplotlib` i `bokeh`. Przykład grafiki `matplotlib` osadzonej w Jupyter Notebook jest pokazany w dolnej części rysunku 1-1. Jupyter/IPython to jedna z najbardziej wyróżniających się historii sukcesu w świecie Pythona.

MicroPython

Utrzymujący się trend miniaturyzacji sprawił, że Python znalazł się w kręgu zainteresowań hobbystów. Komputery jednopłytkowe, takie jak Raspberry Pi (*https://www.raspberrypi.org*) czy Beagle (*https://beagleboard.org*) pozwalają uruchamiać Pythona w pełnym środowisku Linuxa. Poniżej tego poziomu znajdziemy klasę urządzeń nazywanych *mikrokontrolerami* – programowalnych chipów z konfigurowalnym osprzętem – które rozszerzają zakres projektów hobbystycznych i profesjonalnych. Mikrokontrolery na przykład pozwalają odczytywać dane z czujników analogowych i cyfrowych, co umożliwia realizację takich zadań, jak pomiary oświetlenia lub temperatury z minimalną ilością dodatkowego sprzętu.

Zarówno hobbyści, jak i profesjonalni inżynierowie coraz szerzej wykorzystują te urządzenia, które nieustannie pojawiają się na rynku (a niekiedy znikają). Dzięki projektowi MicroPython (*https://micropython.org*) bogata funkcjonalność wielu takich urządzeń (micro:bit, Arduino, pyboard, LEGO® MINDSTORMS® EV3, HiFive itp. – *https://oreil.ly/6Ifug*) może być teraz programowana w Pythonie (a ściślej, w limitowanych dialektach tego języka). W chwili pisania tych słów godne uwagi było wprowadzenie Raspberry Pi Pico (*https://oreil.ly/6-s7Q*). Biorąc pod uwagę sukces Raspberry Pi w świecie edukacyjnym oraz możliwość uruchamiania MicroPythona na Pico wydaje się, że Python dalej konsoliduje swoją pozycję jako języka programowania o największym zakresie zastosowań.

3 Fragmenty te mogą być napisane w różnych językach programowania, nie tylko w Pythonie.

Moving Window Average

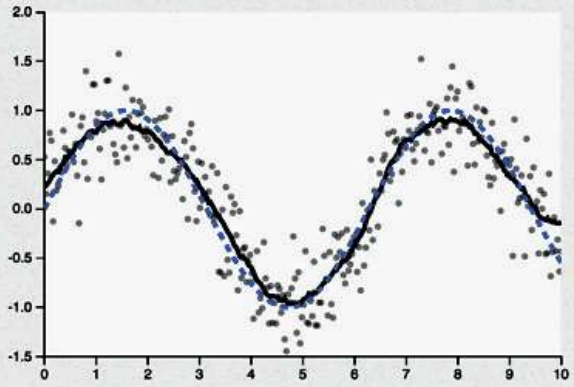
In [6]:
```
np.random.seed(0)
t = np.linspace(0, 10, 300)
x = np.sin(t)
dx = np.random.normal(0, 0.3, 300)

kernel = np.ones(25) / 25.
x_smooth = np.convolve(x + dx, kernel, mode='same')

fig, ax = plt.subplots()
ax.plot(t, x + dx, linestyle='', marker='o',
        color='black', markersize=3, alpha=0.3)
ax.plot(t, x_smooth, '-k', lw=3)
ax.plot(t, x, '--k', lw=3, color='blue')

display_d3(fig)
```

Out[6]:



Rysunek 1-1 *Przykład Jupyter Notebook z osadzonym wykresem utworzonym przez matplotlib*

MicroPython jest implementacją Pythona 3.4 („z wybranymi funkcjonalnościami późniejszych wersji", jak czytamy w dokumentacji (*https://oreil.ly/Xe5YP*)), tworzącym kod bajtowy albo wykonywalny kod maszynowy (większość użytkowników może być spokojnie nieświadoma tego ostatniego faktu). W pełni implementuje składnię Pythona 3.4, ale brakuje w nim większości biblioteki standardowej. Specjalne moduły sterowników sprzętu pozwalają kontrolować różne części wbudowanych urządzeń; dostęp do biblioteki socket Pythona pozwala urządzeniom komunikować się z usługami sieciowymi. Kod może być wywoływany przez zdarzenia zewnętrznych urządzeń i zegara. Dzięki MicroPythonowi język Python może teraz w pełni uczestniczyć w Internecie rzeczy (*Internet of Things* – IoT).

Urządzenie typowo oferuje dostęp do interpretera poprzez szeregowy port USB albo przeglądarkę z wykorzystaniem protokołu WebREPL (*https://oreil.ly/sch3F*). Nic nie jest nam wiadomo o jakiejkolwiek w pełni działającej implementacji `ssh` do tej pory, zatem trzeba zadbać o właściwe zabezpieczenie takich urządzeń: *nie powinny być bezpośrednio dostępne z Internetu bez właściwych, silnych zabezpieczeń!*. Można też zaprogramować sekwencję startową urządzenia po uruchomieniu w Pythonie, tworząc plik *boot.py* w pamięci urządzenia. Plik ten może wykonywać kod MicroPythona o dowolnej złożoności.

Anaconda i Miniconda

Jedną z najbardziej udanych dystrybucji Pythona[4] w ostatnich latach jest Anaconda (*https://www.anaconda.com*). Ten pakiet open source dostarczany jest z wielką liczbą[5] wstępnie skonfigurowanych i przetestowanych modułów rozszerzeń jako uzupełnieniem biblioteki standardowej. W wielu przypadkach może się okazać, że zawiera wszystkie potrzebne zależności dla naszej pracy. Jeśli zależności dla konkretnego projektu nie są obsługiwane, można nadal instalować moduły za pomocą `pip`. W systemach opartych na Unix instaluje się bardzo prosto w pojedynczym katalogu: aby ją aktywować, wystarczy dodać podkatalog *bin* Anacondy na początku zmiennej `PATH` powłoki.

Anaconda bazuje na technologii pakowania o nazwie `conda`. Siostrzana implementacja Miniconda (*https://oreil.ly/dfX4M*) zapewnia dostęp do tych samych rozszerzeń, ale nie ma ich wstępnie załadowanych; zamiast tego pobiera je w razie potrzeby, co czyni ją lepszym wyborem przy tworzeniu dostosowanych środowisk. `conda` nie używa standardowych środowisk wirtualnych, ale zawiera równoważne funkcjonalności, które pozwalają na separację zależności wielu projektów.

pyenv: Proste wsparcie dla wielu wersji

Podstawowym celem `pyenv` (*https://oreil.ly/88o8b*) jest ułatwienie dostępu do tak wielu różnych wersji Pythona, ilu potrzebujemy. Realizuje to, instalując tak zwane skrypty podkładek (*shim*) dla każdego pliku wykonywalnego, które dynamicznie obliczają wymaganą wersję, sprawdzając różne źródła informacji w następującej kolejności:

1. Zmienna środowiskowa `PYENV_VERSION` (jeśli jest ustawiona).
2. Plik *.pyenv_version* w bieżącym katalogu (jeśli istnieje) – można go utworzyć poleceniem **`pyenv local`**.
3. Pierwszy plik *.pyenv_version* odnaleziony podczas przeszukiwania drzewa katalogów w górę (o ile istnieje).
4. Plik *version* w głównym katalogu instalacji `pyenv` – można go utworzyć poleceniem **`pyenv global`**.

`pyenv` instaluje swoje interpretery Pythona poniżej swojego katalogu głównego (którym normalnie jest *~/.pyenv*), a gdy już jest dostępny, wybrany interpreter może zostać skonfigurowany jako domyślny Python w katalogu dowolnego projektu. Alternatywnie (np. przy testowaniu kodu dla wielu wersji) można użyć skryptów do dynamicznego zmieniania interpretera w miarę postępów skryptu.

Polecenie **`pyenv install -list`** pokazuje robiącą wrażenie listę ponad 500 obsługiwanych dystrybucji, w tym PyPy, Miniconda, MicroPython i wiele innych, plus oczywiście

4 W istocie funkcjonalności `conda` obejmują też inne języki, a Python jest po prostu jednym z nich.

5 Ponad 250 automatycznie instalowanych wraz z pakietem Anaconda, przeszło 7500 jawnie instalowalnych przy użyciu **`conda install`**.

każdą oficjalną implementację CPython od wersji 2.1.3 do 3.11.0rc1 (w chwili pisania tych słów).

Transcrypt: Konwertowanie kodu Pythona na JavaScript

Podejmowano wiele prób, aby uczynić Pythona językiem działającym w przeglądarce, ale JavaScript nieustępliwie broni swojego terytorium. System Transcrypt (*https://www.transcrypt.org*) to instalowalny przez `pip` pakiet Pythona, który konwertuje kod Pythona (do wersji 3.9 według stanu aktualnego w chwili pisania tych słów) na wykonywalny w przeglądarce kod JavaScript. Mam pełny dostęp do modelu DOM przeglądarki, co pozwala kodowi na dynamiczne manipulacje zawartością okna i używanie bibliotek JavaScript.

Choć tworzy kod zminimalizowany, Transcrypt udostępnia pełne mapy źródłowe (*https://oreil.ly/WjVAa*), co pozwala na debugowanie z odniesieniami do źródłowego kodu Pythona, a nie generowanego JavaScriptu. Można na przykład pisać procedury obsługi zdarzeń przeglądarki w Pythonie, swobodnie mieszając je z HTML i JavaScript. Być może Python nigdy nie zastąpi JavaScript jako wbudowanego języka przeglądarki, ale Transcrypt sprawia, że nie musimy się już więcej tym martwić.

Inny bardzo aktywny projekt, który pozwala skryptować strony web w Pythonie (do wersji 3.10) to Brython (*https://brython.info*), ale istnieją jeszcze inne: Skulpt (*https://skulpt.org*), obecnie jeszcze nie całkiem zgodny z Python 3, ale zmierzający w tym kierunku; PyPy.js (*https://pypyjs.org*), analogicznie; Pyodide (*https://oreil.ly/jb_US*), obecnie wspierający Python 3.10 i wiele rozszerzeń naukowych, skoncentrowany na koncepcjach Wasm (*https://webassembly.org*). Z ostatnich dodatków wymienić trzeba PyScript (*https://pyscript.net*) Anacondy, zbudowany na bazie Pyodide. Wiele z tych projektów omówimy bardziej szczegółowo w punkcie „Uruchamianie kodu Pythona w przeglądarce" na stronie 35.

Zagadnienia licencjonowania i ceny

CPython jest objęty licencją Python Software Foundation License Version 2 (*https:// oreil.ly/NjjDu*), która jest kompatybilna z GNU Public License (GPL), ale pozwala używać Pythona w wytwarzaniu dowolnego oprogramowania własnościowego, bezpłatnego lub open source, podobnie jak licencje BSD/Apache/MIT. Licencje dla PyPy i innych implementacji są podobnie liberalne. Wszystko, co pobieramy z głównych witryn Pythona i PyPy, nie kosztuje ani grosza. Co więcej, licencje te nie narzucają ograniczeń na warunki licencyjne i cenowe, których można używać dla oprogramowania utworzonego za pomocą obejmowanych przez nie narzędzi, bibliotek i dokumentacji.

Nie wszystko jednak, co dotyczy Pythona, jest wolne od kosztów licencyjnych lub ograniczeń. Wiele źródłowych kodów Pythona, narzędzi i modułów rozszerzeń innych firm ma licencje liberalne, podobne do tej samego Pythona. Inne podlegają licencjom GPL lub Lesser GPL (LGPL), ograniczające warunki, jakie można nałożyć na wytworzone przez

nas prace. Inne komercyjnie opracowane moduły i narzędzia mogą wymagać wniesienia opłat, albo bezwarunkowo, albo wtedy, gdy zostaną użyte w celach zarobkowych[6].

Nic nie może zastąpić starannego zbadania warunków licencji i cen. Zanim zainwestujemy czas i energię w dowolne narzędzie lub komponent programistyczny, trzeba sprawdzić, czy możemy się pogodzić z jego licencją. Często, szczególnie w środowiskach korporacyjnych, takie zagadnienia prawne mogą wymagać konsultacji w prawnikami. Moduły i narzędzia omawiane w tej książce są (o ile nie zostanie jawnie zaznaczone, że jest inaczej) dostępne do pobrania, open source i podlegają liberalnym licencjom podobnym do samego Pythona. Musimy jednak się zastrzec, że nie jesteśmy ekspertami w dziedzinach prawnych, a licencje mogą się zmieniać z czasem, zatem staranne sprawdzenie jest zawsze zalecane.

Powstawanie Pythona i wersje

Python jest opracowywany, utrzymywany i wydawany przez zespół kluczowych programistów, na których czele stoi Guido van Rossum, wynalazca Pythona, architekt, a obecnie „ex" *Benevolent Dictator for Life* (BDFL)[7]. Ten tytuł oznacza, że Guido miał ostatnie słowo w decydowaniu, co ma się stać częścią języka Python i biblioteki standardowej. Od kiedy Guido zdecydował się na wycofanie się z roli BDFL, jego rola podejmowania decyzji została przejęta przez niewielką „Steering Council" (radę kierowniczą), wybieraną na roczne kadencje przez członków PSF.

Własność intelektualna Pythona należy do PSF, korporacji non-profit zajmującej się promowaniem Pythona, opisanej w punkcie „Python Software Foundation" na stronie 16. Wielu PSF Fellows i zwykłych członków ma uprawnienia zatwierdzania w referencyjnych repozytoriach źródeł Pythona (*https://github.com/python*), co jest udokumentowane w „Python Developer's Guide" (*https://oreil.ly/WKjXc*), i większość zatwierdzających to zwykli członkowie albo Fellows w PSF.

Proponowane zmiany w Pythonie są formułowane szczegółowo w publicznych dokumentach określanych mianem Python Enhancement Proposals (PEP) (*https://oreil.ly/HxHfs*). Dokumenty PEP są dyskutowane przez deweloperów Pythona i szerszą społeczność i ostatecznie akceptowane lub odrzucane przez Steering Council (rada może brać pod uwagę przebieg debaty i wstępne głosy, ale nie jest nimi związana). Setki ludzi mają wkład w rozwój Pythona poprzez dokumenty PEP, dyskusje, raporty błędów lub poprawki do źródeł Pythona, bibliotek i dokumentacji.

Podstawowy zespół Pythona wydaje pośrednie wersje Pythona (3.*x* dla rosnących wartości *x*), znanych również jako „feature releases" (wydania funkcjonalne), obecnie w tempie jednego na rok (*https://oreil.ly/VYX-k*).

6 Popularnym modelem biznesowym w tej dziedzinie jest *freemium*: wydawanie zarówno wersji bezpłatnej, jak i komercyjnej wersji „premium" ze wsparciem technicznym, a być może dodatkową funkcjonalnością.

7 „Życzliwy dyktator na całe życie" (przyp. tłum.).

Każde z pośrednich wydań (w odróżnieniu od mikrowydań z poprawkami) dodaje nowe funkcjonalności, które zwiększają siłę Pythona, ale również dbają o zachowanie wstecznej kompatybilności. Python 3.0, w którym dopuszczono złamanie tej wstecznej zgodności w celu usunięcia nadmiarowych „przestarzałych" funkcji i uproszczenia języka, został wydany w grudniu 2008 roku. Python 3.11 (najnowsza wersja stabilna w chwili publikacji tej książki) został opublikowany w październiku 2022 roku.

Każde pośrednie wydanie 3.*x* jest najpierw udostępniane w wersjach alfa, oznakowanej odpowiednio 3.*x*a0, 3.*x*a1 i tak dalej. Po alfach następuje przynajmniej jedno wydanie beta oznaczone 3.*x*b1, a po nim (lub po nich w przypadku większej liczby wersji beta) publikowane jest co najmniej jedno wydanie kandydujące (*release candidate* – RC), 3.*x*rc1. Do czasu opublikowania finalnego wydania wersji 3.*x* (3.*x*.0) jest ono solidne, niezawodne i przetestowane na wszystkich głównych platformach. Każdy programista Pythona może pomóc w zagwarantowaniu tego, pobierając wersje alfa, beta i RC, wypróbowując je i wypełniając raporty o błędach dla dowolnych dostrzeżonych problemów.

Gdy wydanie pośrednie zostanie ukończone, część uwagi podstawowego zespołu kieruje się na kolejne wydanie pośrednie. Jednak wydania pośrednie zwykle uzyskują kolejne wydania podrzędne (czyli 3.*x*.1, 3.*x*.2 i tak dalej), zwykle co dwa miesiące, które nie dodają nowych funkcjonalności, ale mogą poprawiać błędy, rozwiązywać problemy zabezpieczeń, wprowadzać Pythona na nowe platformy, ulepszać dokumentację i dodawać narzędzia oraz (w 100% kompatybilne wstecznie!) optymalizacje.

Wsteczna kompatybilność Pythona jest dość dobra w obrębie wydań głównych. Kod i dokumentacja dla wszystkich starszych wydań Pythona są dostępne online (*https://oreil.ly/JbCv3*), zaś w Dodatku zamieściliśmy podsumowujące listy zmian w każdym wydaniu omawianych w tej książce.

Zasoby dotyczące Pythona

Najbogatszym zasobem źródłowym Pythona jest sieć web, poczynając od własnej witryny Pythona (*https://www.python.org*), pełnej linków do eksploracji.

Dokumentacja

Zarówno CPython, jak i PyPy dostarczane są z dobrą dokumentacją. Podręczniki CPython można czytać online (*https://docs.python.org/3*) (często odwołujemy się do nich jako „dokumentacji online"), ale dostępne są też różne formaty do pobrania, umożliwiające przeglądanie offline, przeszukiwanie i druk. Strona dokumentacji Pythona (*https://www.python.org/doc*) zawiera dodatkowe wskaźniki do ogromnego bogactwa innych dokumentów. Istnieje też strona dokumentacji dla PyPy (*http://doc.pypy.org*). Na koniec dostępne są FAQ online dla Pythona (*https://oreil.ly/-NU8p*) i PyPy (*https://oreil.ly/ajNWC*).

Dokumentacja Pythona dla nieprogramistów

Większość dokumentacji Pythona (w tym ta książka) zakłada pewną wiedzę o wytwarzaniu oprogramowania. Jednak Python doskonale nadaje się również dla osób zaczynających naukę programowania, zatem istnieją wyjątki od tej reguły. Do dobrych wprowadzających (i bezpłatnych) tekstów online dla nieprogramistów należą:

- „Non-Programmers Tutorial for Python 3" autorstwa Josha Cogliati (*https://oreil.ly/HnXMA*) (obecnie skoncentrowany na wersji Python 3.9)
- „Learning to Program" (*https://oreil.ly/FQExV*) Alana Gaulda (obecnie skoncentrowana na Pythonie 3.6)
- *Think Python*, 2 wydanie (*https://oreil.ly/kg6Yd*) Allena Downey'a (skoncentrowana na nieokreślonej wersji Python 3.*x*)

Doskonałym źródłem nauki Pythona (dla nieprogramistów, ale i dla mniej doświadczonych programistów) jest strona wiki „Beginners' Guide to Python" (*https://oreil.ly/Yf5cK*), która zawiera bogactwo linków i porad. Jest tworzona przez społeczność, zatem zachowuje aktualność, gdy dostępne książki, kursy, narzędzia i tak dalej ewoluują i są ulepszane.

Moduły rozszerzeń i źródła

Dobrym punktem wyjścia do poznawania binariów i kodu źródłowego rozszerzeń Pythona jest Python Package Index (*https://oreil.ly/PGIim*) (wciąż czule nazywany „The Cheese Shop" przez niektórych starych wyjadaczy, ale w ogólności określany obecnie jako PyPI), który w chwili pisania tych słów udostępnia ponad 400 000 pakietów, każdy z opisem i wskazówkami.

Standardowa dystrybucja źródłowa zawiera doskonały kod źródłowy Pythona w bibliotece standardowej i katalogu *Tools*, a także źródła w języku C dla wielu wbudowanych modułów rozszerzeń. Nawet jeśli ktoś nie jest zainteresowany budowaniem Pythona ze źródeł, sugerujemy pobranie i rozpakowanie dystrybucji źródłowej (np. ostatniego stabilnego wydania Python 3.11 (*https://oreil.ly/rqYZ9*)) wyłącznie w celu jej przestudiowania. Jeśli ktoś tak postanowi, może też zapoznać się online z najnowocześniejszą wersją biblioteki standardowej Pythona (*https://oreil.ly/zDQ1Z*).

Wiele modułów i narzędzi omawianych w tej książce ma własne, dedykowane im strony. Odsyłacze do takich stron dołączyliśmy do odpowiednich rozdziałów.

Książki

Choć sieć web jest przebogatym źródłem informacji, książki nadal mają swoje miejsce (gdyby tak nie było, nie napisalibyśmy tej książki i nie byłoby czytelników). Powstało mnóstwo książek na temat Pythona. Oto kilka, które zalecamy (choć niektóre omawiają starsze wersje Python 3, a nie aktualne):

- Jeśli ktoś ma już doświadczenie w programowaniu, ale dopiero zaczyna uczyć się Pythona i lubi graficzne podejście w nauce, *Head First Python*, 2 wydanie Paula Barry (O'Reilly) może się dobrze sprawdzić[8]. Podobnie jak wszystkie inne książki z serii *Head First*, wykorzystuje ilustracje i humor do przekazania wiedzy.
- *Dive Into Python 3* (*https://diveintopython3.net*) autorstwa Marka Pilgrima (Apress) uczy na przykładach, zapewniając szybką, ale staranną naukę, odpowiednią dla osób, które już są ekspertami w programowaniu w innych językach.
- *Beginning Python: From Novice to Professional* (*https://oreil.ly/YtWRs*) Magnusa Lie Hetlanda (Apress) uczy zarówno poprzez staranne wyjaśnienia, jak i pełne tworzenie kompletnych programów w różnych obszarach zastosowań.
- *Fluent Python* autorstwa Luciano Ramalho (O'Reilly) to doskonała książka dla bardziej doświadczonych deweloperów, którzy chcą używać bardziej pythonicznych idiomów i funkcjonalności[9].

Społeczność

Jedną z najmocniejszych stron Pythona jest jego solidna, przyjazna i akceptująca społeczność. Programiści i współtwórcy Pythona spotykają się na konferencjach, „hackatonach" (często nazywanych *sprintami* (*https://oreil.ly/oQceG*) w społeczności Pythona) i w lokalnych grupach użytkowników. W społeczności tej aktywnie dyskutowane są wspólne zainteresowania oraz udzielana jest wzajemna pomoc poprzez listy mailingowe i media społecznościowe. Wyczerpującą listę sposobów na dołączenie można znaleźć na stronie *https://www.python.org/community*.

Python Software Foundation

Poza posiadaniem praw do własności intelektualnych języka programowania Python, PSF promuje społeczność Pythona. Poza innymi działaniami, wspiera grupy użytkowników, konferencje i sprinty, a także oferuje granty na rozwój, pomoc i edukację. PSF zawiera dziesiątki uznanych członków nazywanych Fellows (*https://oreil.ly/maILY*) (nominowanych za ich wkład w rozwój Pythona; należy do nich cały podstawowy zespół deweloperski Pythona, a także troje z autorów tej książki); setki członków, którzy poświęcają swój czas, pracę i pieniądze (w tym wielu laureatów Community Service Awards (*https://oreil.ly/MiQRf*)); a także dziesiątki sponsorów korporacyjnych (*https://oreil.ly/FFOZ7*). Każdy, kto używa i wspiera Pythona, może zostać członkiem PSF[10]. Warto sprawdzić stronę o warunkach członkostwa (*https://oreil.ly/MzdRK*) pod kątem informacji o różnych poziomach członkostwa i jak zostać członkiem PSF. Jeśli ktoś jest zainteresowany

8 Wydanie polskie: *Python. Rusz głową*!, Helion 2020.

9 Wydanie polskie: *Zaawansowany Python*, APN Promise 2022.

10 Python Software Foundation utrzymuje znaczącą infrastrukturę wspomagającą ekosystem Pythona. Darowizny dla PSF są zawsze mile widziane.

wniesieniem wkładu w samego Pythona, powinien zapoznać się z „Python Developer's Guide" (*https://oreil.ly/1Jwwb*).

Grupy robocze

Grupy robocze (*https://oreil.ly/0GmfI*) to komitety tworzone przez PSF w celu realizacji konkretnych, ważnych projektów. Oto kilka przykładów grup roboczych aktywnych w czasie pisania tej książki:

- Python Packaging Authority (PyPA) (*https://oreil.ly/0Zxm7*) usprawnia i utrzymuje ekosystem pakietów Pythona oraz publikuje „Python Packaging User Guide" (*https://packaging.python.org*).
- Grupa Python Education (*https://oreil.ly/ZljIc*) promuje edukację i nauczanie przy użyciu Pythona.
- Grupa Diversity and Inclusion (*https://oreil.ly/koEo4*) wspiera i ułatwia rozwój zróżnicowanej i międzynarodowej społeczności programistów Pythona.

Konferencje

Na świecie odbywa się wiele konferencji poświęconych Pythonowi. Ogólne konferencje obejmują wydarzenia międzynarodowe i regionalne, takie jak PyCon (*https://us.pycon.org*) i EuroPython (*https://oreil.ly/nF74d*), ale są też bardziej lokalne, takie jak PyOhio (*http://www.pyohio.org*) lub PyCon Italia (*https://www.pycon.it/en*). Tematyczne konferencje obejmują SciPy (*https://www.scipy2022.scipy.org*) i PyData (*http://pydata.org/ events.html*). Konferencjom często towarzyszą sprinty kodowania, w którym Pythoniści zbierają się razem na kilka dni w celu kodowania określonego projektu open i wzmacniania koleżeństwa. Listę konferencji można znaleźć na stronie *Community Conferences and Workshops* (*https://oreil.ly/asosj*). Ponad 17000 nagrań z wykładów na temat Pythona z ponad 450 konferencji dostępnych jest na stronie PyVideo (*https://pyvideo.org*).

W przypadku znalezienia jakiegoś błędu lub brakującej funkcjonalności w takich modułach otwórz zgłoszenie buga lub funkcji (a najlepiej dostarcz łatkę albo *pull request* naprawiający problem i spełniający potrzeby twojej aplikacji). Innymi słowy, zachęcamy, abyś stał(a) się aktywnym członkiem społeczności open source, a nie jedynie pasywnym użytkownikiem: będziesz tam mile widziana/y, połechtasz swoje własne ego, a pewnie pomożesz wielu innym. „Podaj dalej", gdy nie możesz „oddać" tym wszystkim wspaniałym ludziom, którzy przyczynili się do powstania niemal wszystkich narzędzi, których używasz!

Grupy użytkowników i organizacji

Społeczność Pythona dysponuje lokalnymi grupami użytkowników na każdym kontynencie poza Antarktyką[11] – jest ich ponad 1600, zgodnie z listą dostępną w wiki LocalUserGroups (*https://oreil.ly/cY6Mk*). Na całym świecie odbywają się też meetupy na temat Pythona (*https://oreil.ly/h6oEs*). PyLadies (*http://www.pyladies.com*) to międzynarodowa grupa mentoringu z lokalnymi oddziałami promująca udział w tworzeniu i wykorzystywania Pythona; wszyscy zainteresowani Pythonem są mile widziani. NumFOCUS (*https://numfocus.org*), organizacja dobroczynna nonprofit promująca otwarte podejście do badań, danych i informatyki wspiera konferencje PyData i inne projekty.

Listy mailingowe

Strona Community Mailing Lists (*https://www.python.org/community/lists*) zawiera linki do wielu list mailingowych dotyczących Pythona (i do kilku grup Usenetu, dla tych, którzy są dość starzy, aby pamiętać Usenet (*https://oreil.ly/5qYdq*)!). Alternatywnie można przeszukać serwis Mailman (*https://mail.python.org/archives*), aby znaleźć aktywne listy mailingowe obejmujące szeroki zakres tematów. Oficjalne powiadomienia dotyczące Pythona są publikowane na liście python-announce (*https://oreil.ly/eg9Ft*). Szukając pomocy w konkretnych problemach można napisać na adres *help@python.org*. Pomoc w zakresie nauki lub uczenia Pythona można uzyskać pod adresem *tutor@python.org*, albo, jeszcze lepiej, dołączyć do listy (*https://oreil.ly/iEQJF*). Użyteczne tygodniowe podsumowanie wiadomości i artykułów dotyczących Pythona można uzyskać, subskrybując stronę Python Weekly (*http://www.pythonweekly.com*). Można też śledzić Python Weekly pod *@python_discussions@mastodon.social*.

Media społecznościowe

Feed RSS (*https://oreil.ly/pf4AS*) blogów dotyczących Pythona udostępnia strona Planet Python (*http://planetpython.org*). Osoby zainteresowane śledzeniem rozwoju języka powinny sprawdzić stronę *discuss.python.org* – wysyła ona użyteczne podsumowania, jeśli ktoś nie odwiedza jej regularnie. Na Twitterze warto śledzić konto @ThePSF. Platforma Libera.Chat (*https://libera.chat*) na IRC (*https://oreil.ly/AXMAf*) hostuje kilka kanałów związanych z Pythonem: głównym z nich jest `#python`. LinkedIn (*https://www.linkedin.com*) zawiera wiele grup pythonistów, w tym Python Web Developers (*https://oreil.ly/--LKFZ*). W witrynie Slack można dołączyć do społeczności PySlackers (*https://pyslackers.com*). W serwisie Discord warto sprawdzić Python Discord (*https://pythondiscord.com*). Pytania techniczne i odpowiedzi na temat programowania w Pythonie można też znaleźć i śledzić w witrynie Stack Overflow (*http://stackoverflow.com*) pod wieloma różnymi tagami, w tym [python] (*https://oreil.ly/GHoVY*). Python jest obecnie najbardziej aktywnym językiem programowania na Stack Overflow (*https://oreil.ly/K3oK3*) i można tam

11 Chyba powinniśmy się zmobilizować, aby więcej pingwinów zainteresowało się naszym językiem!

znaleźć wiele przydatnych odpowiedzi wraz z olśniewającymi dyskusjami. Jeśli zaś ktoś lubi podcasty, warto sprawdzić podcasty Pythona, Python Bytes (*https://pythonbytes.fm*).

Instalacja

Wersję klasyczną (CPython) oraz PyPy Pythona można zainstalować na większości platform. Mając odpowiedni system deweloperski (C dla CPython; PyPy jako zakodowany w samym Pythonie wymaga wcześniejszego zainstalowania CPython), można zainstalować wersje Pythona z odpowiednich dystrybucji kodu źródłowego. W przypadku popularnych platform dostępna jest też (zalecana) alternatywa w postaci instalacji wstępnie zbudowanych dystrybucji binarnych.

Instalowanie Pythona, jeśli jest wstępnie zainstalowany

Jeśli używana platforma dostarczana jest z wstępnie zainstalowaną wersją Pythona, nadal najlepszym wyborem jest zainstalowanie oddzielnej, aktualnej wersji na potrzeby tworzenia własnego kodu. W takim przypadku *nie* należy usuwać ani nadpisywać oryginalnej wersji platformy: zamiast tego instalujemy nową wersję obok pierwszej. W ten sposób nie zakłócimy działania żadnego innego oprogramowania, które jest częścią platformy: takie oprogramowanie może bazować na konkretnej wersji Pythona dostarczonej z samą platformą.

Instalowanie CPython z dystrybucji binarnej jest szybsze, oszczędza sporo pracy na niektórych platformach i jest jedyną możliwością, jeśli nie dysponujemy odpowiednim kompilatorem języka C.

Instalowanie z kodu źródłowego zapewnia większą kontrolę i elastyczność i jest konieczne, jeśli nie uda się znaleźć pasującej dystrybucji binarnej dla naszej platformy. Nawet przy instalowaniu z binariów warto pobrać dystrybucję źródłową, gdyż może zawierać przykłady, dema i narzędzia, których zwykle brakuje we wstępnie zbudowanych binariach. Przyjrzyjmy się teraz, jak zrobić jedno i drugie.

Instalowanie Pythona z plików binarnych

Jeśli używana platforma jest popularna i aktualna, można łatwo znaleźć wstępnie skompilowane, spakowane binarne wersje Pythona gotowe do instalacji. Pakiety binarne są typowo samoinstalujące, albo bezpośrednio jako programy wykonywalne, albo za pośrednictwem odpowiednich narzędzi systemowych, takich jak Red Hat Package Manager (RPM) w niektórych wersjach Linuxa lub Microsoft Installer (MSI) w Windows. Po pobraniu pakietu instalujemy go, uruchamiając program i wybierając parametry instalacyjne, takie jak katalog, w którym Python ma być zainstalowany. W Windows trzeba zaznaczyć opcję opisaną „Add Python 3.10 to PATH", aby nakazać instalatorowi dodanie

lokalizacji instalacyjnej do zmiennej środowiskowej PATH w celu ułatwienia posługiwania się Pythonem w wierszu polecenia (patrz „Program python” na stronie 25).

„Oficjalne” binaria można uzyskać ze strony Downloads (*https://oreil.ly/b3AP7*) witryny Pythona: kliknięcie przycisku opisanego „Download Python 3.11.x” powoduje pobranie najbardziej aktualnej binarnej wersji, odpowiedniej do platformy zgłaszanej przez przeglądarkę.

Wiele innych źródeł udostępnia bezpłatnie binarne instalatory Pythona dla innych platform. Istnieją instalatory powiązane z dystrybucjami Linuxa, zarówno tych bazujących na RPM (*http://rpmfind.net*) (Red Hat, Fedora, Mandriva, SUSE itp.), jak i na Debianie (*http:// www.debian.org*) (do tej grupy zalicza się Ubuntu, zapewne najbardziej popularna dystrybucja Linuxa w chwili pisania tych słów). Strona Other Platforms (*https://oreil.ly/xvFYV*) zawiera linki do dystrybucji binarnych dla nawet dość egzotycznych platform, takich jak AIX, OS/2, RISC OS, IBM AS/400, Solaris, HP-UX i tak dalej (choć często nie są to najnowsze wersje Python, co jednak nie powinno dziwić przy obecnie „osobliwej” naturze tych platform), a także dla najbardziej aktualnej wersji iOS (*https://oreil.ly/gnJND*), systemu operacyjnego dla popularnych urządzeń iPhone i iPad.

Wspomniana wcześniej w tym rozdziale Anaconda (*https://oreil.ly/DxmAG*) jest binarną dystrybucją zawierającą Pythona plus menedżera pakietów `conda` (*http://conda.pydata.org/docs*), a do tego setki rozszerzeń innych firm, w szczególności tych przeznaczonych do prac naukowych, matematycznych, inżynierskich i analizy danych. Anaconda jest dostępna dla systemów Linux, Windows i macOS. Miniconda (*https://oreil.ly/RrY5_*), również wspomniana wcześniej, to ten sam pakiet, ale bez tych wszystkich rozszerzeń; można wybiórczo instalować ich podzbiory przy użyciu `conda`.

macOS

Popularny niezależny menedżer pakietów open source dla macOS, Homebrew (*http://brew.sh*), oferuje poza wieloma innymi pakietami doskonałe wersje Pythona (*https:// oreil.ly/rnK6U*). `conda`, wspomniana w punkcie „Anaconda i Miniconda” na stronie 11, również działa doskonale w systemie macOS.

Instalowanie Pythona z kodu źródłowego

Do zainstalowania CPython z kodu źródłowego potrzebujemy platformy dysponującej kompilatorem C zgodnym z ISO oraz narzędziami takimi jak `make`. W systemie Windows normalny sposób zbudowania Pythona polega na użyciu Visual Studio (ideally VS 2022 (*https://oreil.ly/eblTI*), który obecnie jest dostępny bez opłat (*https://oreil.ly/2j1dK*)).

W celu pobrania kodu źródłowego Pythona należy odwiedzić stronę Python Source Releases (*https:// oreil.ly/HeGVY*). W witrynie Pythona należy wskazać opcję Downloads w pasku menu i wybrać „Source code”, po czym wybrać odpowiednią wersję.

Plik dostępny pod tym łączem, opisany jako „Gzipped source tarball", ma rozszerzenie *.tgz*; jest to równoważne rozszerzeniu *.tar.gz* (czyli mamy tu archiwum *tar* plików, skompresowane za pomocą popularnego kompresora `gzip`). Alternatywnie można użyć linku opisanego jako „XZ compressed source tarball", aby uzyskać plik z rozszerzeniem *.tar.xz* zamiast *.tgz*, spakowany jeszcze wydajniejszym kompresorem `xz`, o ile dysponujemy wszystkimi narzędziami potrzebnymi do poradzenia sobie z kompresją XZ.

Microsoft Windows

W systemie Windows instalowanie Pythona z kodu źródłowego może być uciążliwe, o ile ktoś nie zna dobrze programu Visual Studio i nie jest przyzwyczajony do pracy w oknie tekstowym znanym jako *wiersz polecenia*[12] – większość użytkowników Windows preferuje po prostu pobranie wstępnie zbudowanego Pythona z Microsoft Store (*https://oreil.ly/wNIMo*).

Jeśli poniższe instrukcje będą sprawiać problemy, lepiej pozostać przy instalowaniu Pythona z plików binarnych, zgodnie z opisem w poprzednim podrozdziale. Najlepsze zalecenie to wykonanie oddzielnej instalacji z binariów, nawet jeśli również instalujemy ze źródeł. Gdy ktoś zauważy coś dziwnego przy używaniu wersji zbudowanej ze źródeł, trzeba to zweryfikować w instalacji binarnej. Jeśli dziwactwo zniknie, musi być to spowodowane jakimś niedociągnięciem w instalacji wykonanej ze źródeł, zatem wiadomo, że trzeba starannie sprawdzić szczegóły wykonania tej ostatniej.

W kolejnych punktach przyjmujemy dla przejrzystości, że został wykonany nowy folder o nazwie *%USERPROFILE%\py* (np. *c:\users\tim\py*), który można utworzyć na przykład wpisując polecenie **`mkdir`** w dowolnym oknie polecenia. Pobieramy źródłowy plik *.tgz* – na przykład *Python-3.11.0.tgz* – do tego folderu. Naturalnie można nazwać i umieścić ten folder w miejscu, który najbardziej nam odpowiada: nasz wybór nazwy ma tylko ułatwić objaśnienia.

Dekompresowanie i rozpakowywanie kodu źródłowego

Plik *.tgz* lub *.tar.xz* można zdekompresować i rozpakować na przykład przy użyciu darmowego programu 7-Zip (*http://www.7-zip.org*). Pobieramy odpowiednią wersję ze strony Download (*https://oreil.ly/Fwv5d*), instalujemy ją i uruchamiamy względem pliku *.tgz* (np. *c:\users\alex\py\Python-3.11.0.tgz*) pobranego z witryny Pythona.

Zakładając, że pobraliśmy ten plik do folderu *%USERPROFILE%\py* (albo przenieśliśmy go tam z folderu *%USERPROFILE%\downloads*), otrzymaliśmy teraz folder o nazwie *%USERPROFILE%\py\Python-3.11.0* lub podobnej, stosownie do pobranej wersji. Jest to korzeń drzewa katalogów, zawierającego pełną standardową dystrybucję Pythona w postaci źródłowej.

12 Albo zdecydowanie preferowanym Windows Terminal w najnowszych wersjach Windows (*https://oreil.ly/_Cu97*).

Budowanie kodu źródłowego Pythona

Otwieramy plik *readme.txt* zlokalizowany w podkatalogu *PCBuild* tego katalogu głównego (w dowolnym edytorze tekstowym), a następujemy zgodnie z zawartymi w nim szczegółowymi instrukcjami.

Platformy unixowe

Na platformach unixowych instalowanie Pythona z kodu źródłowego w ogólności jest proste[13]. W kolejnych punktach przyjmujemy dla przejrzystości, że utworzony został nowy katalog o nazwie *~/py* i został do niego pobrany źródłowy plik *.tgz* – na przykład *Python-3.11.0.tgz*. Oczywiście można nazwać i zlokalizować ten katalog w miejscu, który najlepiej odpowiada potrzebom: nasz wybór nazwy ma tylko ułatwić objaśnienia.

Dekompresowanie i rozpakowywanie kodu źródłowego

Plik *.tgz* lub *.tar.xz* można zdekompresować i rozpakować przy użyciu popularnej wersji GNU `tar`. Wystarczy w powłoce wpisać poniższe polecenie:

```
$ cd ~/py && tar xzf Python-3.11.0.tgz
```

W rezultacie uzyskamy katalog o nazwie *~/py/Python-3.11.0* lub podobnej, zależnie od pobranej wersji. Jest to korzeń drzewa katalogów, zawierającego pełną standardową dystrybucję Pythona w postaci źródłowej.

Konfigurowanie, budowanie i testowanie

Szczegółowe uwagi zawiera plik *README* wewnątrz tego katalogu pod nagłówkiem „Build instructions" i zdecydowanie polecamy przestudiowanie tych uwag. Jednak w najprostszym przypadku wszystko, czego potrzebujemy, to wpisanie poniższych poleceń w powłoce:

```
$ cd ~/py/Python-3.11/0
$ ./configure
    [configure wypisuje bardzo dużo informacji, pominiętych tutaj]
$ make
    [działanie make zajmuje sporo czasu i również emituje wele informacji,
     tu pominiętych]
```

13 Większość problemów związanych z instalowaniem ze źródeł wynika z nieobecności rozmaitych bibliotek pomocniczych, co może spowodować, że pewne funkcjonalności będą nieobecne w zbudowanym interpreterze. „Python Developers' Guide" wyjaśnia, jak obsługiwać zależności na różnych platformach (*https://oreil.ly/j3XJs*). *buildpython-from-source.com* to pomocna witryna, która pokazuje wszystkie polecenia konieczne do pobrania, zbudowania i zainstalowania wybranych wersji Pythona, plus większość wymaganych pomocniczych bibliotek dla wielu platform linuxowych.

Jeśli uruchomimy **make** bez wcześniejszego wywołania **./configure**, **make** niejawnie uruchamia **./config ure**. Po zakończeniu działania **make** sprawdzamy, że zbudowany przez nas Python działa zgodnie z oczekiwaniem:

```
$ make test
    [trwa dość długo, wyświetla dużo informacji pominiętych tutaj]
```

Zazwyczaj **make test** potwierdza, że wykonana kompilacja działa, ale informuje też, że pewne testy zostały pominięte ze względu na nieobecność modułów opcjonalnych.

Niektóre z modułów są specyficzne dla platformy (np. niektóre mogą działać jedynie na maszynach używających antycznego systemu operacyjnego IRIX firmy SGI (*https://oreil.ly/SsGHY*)); tymi nie potrzebujemy się martwić. Jednak inne moduły mogły zostać pominięte, gdyż są zależne od innych pakietów open source, które aktualnie nie są zainstalowane na naszym komputerze. Dla przykładu w Uniksach moduł `_tkinter` – wymagany do uruchomienia pakietu graficznego interfejsu użytkownika Tkinter oraz zintegrowanego środowiska programistycznego IDLE dostarczanego z Pythonen – może zostać skompilowany jedynie wtedy, gdy **./configure** zdoła odnaleźć w maszynie instalację Tcl/Tk 8.0 lub późniejszą. Plik *README* zawiera więcej szczegółów i szczególnych pułapek występujących w różnych platformach uniksowych i uniksopodobnych.

Budowanie z kodu źródłowego pozwala dostosować konfigurację na wiele sposobów. Na przykład można skompilować Pythona w specjalny sposób, który pomaga debugować wycieki pamięci przy projektowaniu rozszerzeń kodowanych w C, co jest omówione w punkcie „Building and Installing C-Coded Python Extensions" w dostępnym online rozdziale 25 (*https://oreil.ly/pythonnutshell-25*). **./configure --help** jest dobrym źródłem informacji na temat możliwych do użycia opcji konfiguracyjnych.

Instalowanie po kompilacji

Domyślnie **./configure** przygotowuje Pythona do instalacji w katalogach */usr/local/bin* i */usr/local/lib*. Można zmienić te ustawienia, uruchamiając **./configure** z opcją **--prefix** przed wywołaniem **make**. Jeśli na przykład potrzebujemy prywatnej instalacji Pythona w podkatalogu *py311* w swoim katalogu domowym, wykonamy:

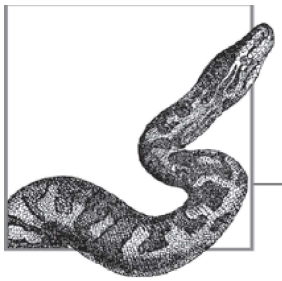
```
$ cd ~/py/Python-3.11.0
$ ./configure --prefix=~/py311
```

po czym przechodzimy do **make**, jak w poprzednim punkcie. Po wykonaniu kompilacji i testowania Pythona możemy wykonać faktyczną instalację wszystkich plików, wykonując poniższe polecenie:[14]

```
$ make install
```

14 Albo **make altinstall**, jeśli chcemy uniknąć tworzenia linków do plików wykonywalnych Pythona i stron podręcznika.

Użytkownik uruchamiający **`make install`** musi mieć uprawnienia do zapisu w docelowych katalogach. Zależnie od wyboru tych katalogów i uprawnień w tych katalogach konieczne może być wykonanie **`su`**, aby przejść do konta *root, bin* lub jakiegoś innego użytkownika o podniesionych uprawnieniach przed wywołaniem **`make install`**. Powszechny idiom dla takich celów to **`sudo make install`**: jeśli **`sudo`** zażąda podania hasła, należy wpisać swoje własne hasło użytkownika, a nie hasło *root*. Alternatywne i zalecane podejście to instalacja w środowisku wirtualnym, omówiona w punkcie „Środowiska Pythona” na stronie 267.



2
Interpreter Pythona

Podczas tworzenia systemów oprogramowania w Pythonie zazwyczaj piszemy pliki tekstowe zawierające kod źródłowy. Można to robić przy użyciu dowolnego edytora tekstowego, w tym tych, które wyliczamy w punkcie „Środowiska deweloperskie Pythona" na stronie 31. Następnie przetwarzamy pliki źródłowe za pomocą kompilatora i interpretera Pythona. Można to robić bezpośrednio wewnątrz zintegrowanego środowiska programistycznego (*integrated development environment* – IDE) albo za pośrednictwem innego programu, który opakowauje Python. Interpreter Pythona również pozwala wykonywać kod Pythona, tak jak IDE.

Program python

Interpreter Pythona jest uruchamiany jako program **`python`** (w systemie Windows nosi nazwę *python.exe*). Program ten zawiera zarówno sam interpreter, jak i kompilator Pythona, który jest wywoływany niejawnie, gdy jest potrzebny dla importowanych modułów. Zależnie od systemu, program może wymagać umieszczenia w katalogu wymienionym w zmiennej środowiskowej `PATH`. Alternatywnie, jak w przypadku każdego innego programu, można podać jego pełną ścieżkę w znaku zachęty (powłoce) albo w skrypcie powłoki (albo w skrócie i tak dalej), który go uruchamia[1].

W systemie Windows należy nacisnąć klawisz Windows i zacząć wpisywać **`python`**. Pojawi się wpis „Python 3.x" (wersja wiersza poleceń) obok innych możliwości, takich jak „IDLE" (czyli graficzny interfejs Pythona).

1 Może to wymagać użycia cudzysłowów, jeśli ścieżka zawiera spacje – ponownie jest to zależne od systemu operacyjnego.

Zmienne środowiskowe

Oprócz `PATH`, jeszcze inne zmienne środowiskowe mają wpływ na działanie programu **`python`**. Niektóre z nich mają takie same efekty, jak opcje przekazywane do **`python`** w wierszu poleceń, które pokażemy w następnym podrozdziale, ale wiele zmiennych udostępnia ustawienia, które nie są dostępne jako opcje wiersza poleceń. Poniższa lista przedstawia kilka z najczęściej używanych; pełny szczegółowy spis dostępny jest w dokumentacji online (*https://oreil.ly/sYdEK*):

`PYTHONHOME`
: Katalog instalacyjny Pythona. Podkatalog *lib*, zawierający bibliotekę standardową Pythona, musi znajdować się w tym katalogu. W systemach uniksowych moduły biblioteki standardowej muszą znajdować się w katalogu *lib/python-3.x* (dla wersji Python 3.*x*, gdzie *x* jest pośrednim numerem wersji). Jeśli zmienna `PYTHONHOME` nie jest ustawiona, Python przyjmuje przypuszczenia na temat katalogu instalacyjnego.

`PYTHONPATH`
: Lista katalogów, w systemach uniksowych oddzielana dwukropkami, a w Windows średnikami, z których Python może importować moduły. Lista ta rozszerza wstępną wartość zmiennej `sys.path` Pythona. Moduły, ich importowanie i zmienną `sys.path` omówimy w rozdziale 7.

`PYTHONSTARTUP`
: Nazwa pliku źródłowego w Pythonie, który ma być wywoływany przy każdym uruchomieniu interaktywnej sesji interpretera. Żaden taki plik nie jest uruchamiany, jeśli nie ustawimy tej zmiennej albo nie zostanie ustawiona na ścieżkę do nieistniejącego pliku. Plik `PYTHONSTARTUP` nie jest uruchamiany, gdy wywołujemy skrypt Pythona; działa tylko przy rozpoczynaniu sesji interaktywnej.

To, jak ustawiać i sprawdzać zmienne środowiskowe, zależy od systemu operacyjnego. W systemach Unix użyjemy poleceń powłoki, często wewnątrz rozruchowych skryptów powłoki. W systemie Windows naciskamy klawisz Windows i zaczynamy wpisywać **`environment var`**[2], po czym pojawi się kilka skrótów: jeden dla zmiennych środowiskowych użytkownika, zaś pozostałe dotyczą systemu. Na komputerach Mac można pracować tak samo, jak w innych systemach uniksowych, ale mamy tu więcej opcji, w tym IDE specyficzne dla MacPython. Więcej informacji na temat używania Pythona na Macu zawiera artykuł „Using Python on a Mac” w dokumentacji online (*https://oreil.ly/Co1au*).

Składnia i opcje wiersza poleceń

Składnię wiersza poleceń interpretera Pythona można podsumować jak poniżej:

```
[ścieżka]python {opcje} [-c polecenie | -m moduł | plik | -] {argumenty}
```

2 W polskiej wersji Windows należy wyszukać ciąg „zmienne środowiskowe” (przyp. tłum.).

Nawiasy kwadratowe ([]) obejmują elementy opcjonalne. Nawiasy klamrowe ({}) obejmują elementy, które mogą mieć zero lub więcej wystąpień, zaś pionowe kreski (|) oznaczają wybór pomiędzy alternatywnymi opcjami. Python używa ukośnika (/) w ścieżkach plików, tak jak standardowo w systemach Unix.

Uruchomienie skryptu Pythona w wierszu poleceń może być tak proste, jak poniższe:

```
$ python hello.py
Hello World
```

Można też jawnie podać ścieżkę do skryptu:

```
$ python ./hello/hello.py
 Hello World
```

Nazwa pliku skryptu może być ścieżką bezwzględną lub względną i nie musi mieć konkretnego rozszerzenia nazwy (choć konwencjonalnie używa się rozszerzenia *.py*). *opcje* to krótkie ciągi z rozróżnianiem wielkości liter, rozpoczynające się dywizem (-), które żądają od programu **python** zachowania innego niż domyślne. **python** akceptuje tylko opcje zaczynające się dywizem. Najczęściej używane opcje zostały wymienione w tabeli 2-1. Opis każdej opcji zawiera zmienną środowiskową (jeśli istnieje), której ustawienie wymusza takie zachowanie. Wiele opcji ma dłuższe wersje rozpoczynane dwoma dywizami, które można zobaczyć poleceniem **python -h**. Pełna lista ze szczegółami dostępna jest w dokumentacji online (*https://oreil.ly/1ZcA9*).

Tabela 2-1 *Często używane opcje wiersza poleceń programu python*

Opcja	Znaczenie (i odpowiadająca zmienna środowiskowa, jeśli istnieje)
-B	Nie zapisuje plików kodu bajtowego (bytecode) na dysku (PYTHONDONTWRITEBYTECODE)
-c	Podaje instrukcje Pythona w wierszu poleceń
-E	Ignoruje wszystkie zmienne środowiskowe
-h	Wyświetla pełną listę opcji i kończy działanie
-i	Uruchamia sesję interaktywną po zakończeniu działania pliku lub polecenia (PYTHONINSPECT)
-m	Specyfikuje moduł Pythona uruchamiany jako główny skrypt
-O	Optymalizuje kod bajtowy (PYTHONOPTIMIZE) – zwróćmy uwagę, że jest to wielka litera O, a nie cyfra 0
-OO	Podobnie jak -O, ale dodatkowo usuwa komentarze docstring z kodu bajtowego
-S	Pomija niejawną stronę importu przy uruchamianiu (szczegółowe omówienie zawiera punkt „Dostosowywanie dla lokacji" na stronie 485)
-t, **-tt**	Zgłasza ostrzeżenia o niespójnym użyciu tabulatorów (-tt generuje błędy, a nie ostrzeżenia, dla tych samych problemów)
-u	Używa niebuforowanych plików binarnych jako wyjścia standardowego i standard wyjścia błędów (PYTHONUNBUFFERED)
-v	Szczegółowo pokazuje importowanie modułów i działania porządkowe (PYTHONVERBOSE)

Tabela 2-1 *Często używane opcje wiersza poleceń programu* `python`

Opcja	Znaczenie (i odpowiadająca zmienna środowiskowa, jeśli istnieje)
`-V`	Wypisuje numer wersji Pythona i kończy działanie
`-W arg`	Dodaje wpis do filtra ostrzeżeń (patrz „Moduł warnings" na stronie 605)
`-x`	Wyklucza (pomija) pierwszy wiersz pliku źródłowego skryptu

Warto nabrać nawyku używania `-i`, aby uzyskać interaktywną sesję natychmiast po wykonaniu jakiegoś skryptu, z dostępnymi do zbadania, nadal nienaruszonymi zmiennymi najwyższego poziomu. Nie ma potrzeby używania `-i` w normalnych sesjach interaktywnych, ale użycie tej opcji też nie zaszkodzi.

`-O` zapewniają `-OO` niewielkie oszczędności czasu i miejsca w kodzie bajtowym dla importowanych modułów, zamieniając instrukcje `assert` na brak operacji, zgodnie z omówieniem zawartym w podrozdziale „Instrukcja assert" na stronie 246. Opcja `-OO` dodatkowo odrzuca ciągi dokumentujące[3].

Po opcjach, o ile zostały użyte, informujemy Python, który skrypt ma uruchomić, podając ścieżkę do tego skryptu. Zamiast ścieżki pliku można użyć `-c` *`polecenie`*, aby wykonać pewien ciąg poleceń Pythona. Parametr *`polecenie`* zazwyczaj zawiera spacje, zatem trzeba dodać wokół niego cudzysłów, aby spełnić wymagania powłoki lub procesora wiersza poleceń systemu operacyjnego. Niektóre powłoki (np. `bash` (*https://oreil.ly/seIne*)) pozwalają wpisać wiele wierszy jako jeden argument, zatem *`polecenie`* może być szeregiem instrukcji Pythona. Inne powłoki (np. Windows) narzucają ograniczenie do pojedynczego wiersza; *`polecenie`* może być jedną instrukcją lub kilkoma prostymi instrukcjami oddzielanymi średnikami (`;`), co omówimy w punkcie „Instrukcje" na stronie 48.

Innym sposobem wyspecyfikowania skryptu do uruchomienia jest użycie opcji `-m` *`moduł`*. Opcja ta nakazuje Pythonowi załadować i uruchomić moduły nazwanego *`moduł`* (albo pliku *__main__.py* w pakiecie lub pliku ZIP o nazwie *`moduł`*) z pewnego katalogu, który jest częścią zmiennej `sys.path` Pythona. Jest to przydatne w przypadku wielu modułów z biblioteki standardowej Pythona. Na przykład, jak wyjaśnimy w podpunkcie „Moduł timeit" na stronie 620, `-m timeit` jest często najlepszym sposobem wykonywania mikropomiarów instrukcji Pythona.

Dywiz (`-`) albo brak jakiegokolwiek tokenu w tej pozycji nakazuje interpreterowi odczytywać kod źródłowy z wejścia standardowego – normalnie będzie to sesja interaktywna. Potrzebujemy użyć dywizu tylko wtedy, gdy następują dalsze argumenty. *Argumenty* są dowolnymi ciągami; uruchomiony interpreter Pythona będzie miał dostęp do tych ciągów jako elementów listy `sys.argv`.

Dla przykładu wpiszemy poniższe polecenie, aby Python pokazał bieżącą datę i czas:

```
$ python -c "import time; print(time.asctime())"
```

3 Może to wpływać na kod analizujący docstringi z uzasadnionych powodów; sugerujemy, aby unikać tworzenia takiego kodu.