Piotr Wróblewski

PYTHON
dla testera

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion S.A.
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/pyttes
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
ftp://ftp.helion.pl/przyklady/pyttes.zip

ISBN: 978-83-283-8404-0

Printed in Poland.

Spis treści

Przedmowa **9**

Rozdział 1. **Czysty start, czyli zapanuj nad instalacjami** **15**
Testujemy poprawność instalacji Pythona 17
Instalator pip i biblioteki Pythona 20
Edytory do Pythona 21
Środowiska IDE (i dlaczego PyCharm) 22
IDLE 22
PyCharm 26
Dokumentacja Pythona 30
Używanie zasobów GitHuba 32
Podsumowanie 33

Rozdział 2. **Praca w linii poleceń** **35**
Wywołanie terminala linii poleceń 36
Zasoby komputera bez tajemnic 38
Drzewo katalogów 38
Polecenia używane do nawigacji po katalogach 39
Tworzenie i kasowanie elementów 42
Uruchamianie programów 42
Wyświetlanie zawartości pliku 43
Porównywanie zawartości plików 44
Przekierowanie wyniku działania skryptu do pliku 45

Rozdział 3. **Niezbędnik** **47**
Zasady formatowania kodu 48
Systemy liczbowe w (strawnej) pigułce 49
Operatory 51
Operatory arytmetyczne 51
Operatory bitowe 51
Operatory logiczne i wyrażenia warunkowe 54
Zmienne 56
Gdzie te typy danych? 56
Pojęcie referencji 57
Zachowaj porządek! 59
Funkcje i metody matematyczne 60
Napisy w Pythonie 61
Notacja f" 63

Kłopotliwy dwukropek, czyli zakresy w Pythonie ... 66
Konwersje napisów na liczby (i odwrotnie) ... 66
Zamiana napisów na listy elementów ... 67
Pętle for i while ... 67
Funkcje i procedury ... 70
Pierwsza funkcja ... 70
Parametry domyślne ... 71
Rekurencja ... 71
Zmienna liczba parametrów to nie problem! ... 73
Zasięg zmiennych ... 74
Notacja z kropką ... 75
Gotowe klasy biblioteczne ... 76

Rozdział 4. **Python z klasą ... 79**
Szablon tworzenia klasy ... 80
Klasy w wersji „PRO” ... 83
Obiekty tworzone w wyniku operacji arytmetycznych ... 88
Dziedziczenie bywa proste ... 90
Podsumowanie praktycznych celów OOP ... 94

Rozdział 5. **Przybornik skryptologa ... 95**
Najpierw pomyśl, potem rób! ... 95
Parametry skryptów ... 96
Parametry w wersji PRO ... 109
Moduły, czyli własne biblioteki ... 100
Scenariusze pod kontrolą ... 101
Interakcja z użytkownikiem ... 103
Kontrola błędów, czyli wyjątki ... 104
Menu sterujące skryptem ... 108
Wywołanie zewnętrznego programu w skrypcie ... 110
Wersja Windows ... 110
Wersja macOS/Linux ... 111
Moduły i pakiety ... 113
Publikacja modułu w Internecie ... 115

Rozdział 6. **Podane na tacy ... 117**
Napisy — podsumowanie ... 118
Listy, czyli... tablice dynamiczne ... 120
Metody dostępne dla list w Pythonie ... 124
Z listy na stos ... 125
Przykład użycia listy ... 126
Listy tworzone na podstawie wyrażeń ... 129
Tuple, czyli „co to za dziwoląg” ... 131
Modyfikacja tupli ... 132
Zastosowania programistyczne ... 133
Zbiory ... 134
Zbiory tworzone na podstawie wyrażeń ... 138
Słowniki ... 139

Rozdział 7. **Magia zaszyta w plikach ... 145**
Podsumowanie kilku pojęć dotyczących systemów plikowych ... 147
Binarnie czy tekstowo? ... 148
Odczyt plików tekstowych ... 149
Zapis danych do plików tekstowych ... 153

Podsumowanie metod odczytu i zapisu plików tekstowych ... 155
Odczyt plików binarnych ... 155
Sposób na nieśmiertelność... danych ... 157
Serializacja obiektów (pickle) ... 158
Operacje na plikach i folderach (moduł os) ... 160
Format ścieżki, czyli kłopotliwy ukośnik ... 161
Usuwanie i tworzenie katalogów ... 162
Ścieżki z klasą... Path ... 162
Podstawowe operacje na obiektach klasy Path ... 162
Pokaż, co tam trzymasz w... folderze! ... 165
Nasi tu byli! ... 168
Exterminate! ... 170
Usuwanie plików lub katalogów ... 170
Przesuwanie plików lub katalogów ... 172
Sztuczki, porady, sugestie... ... 173

Rozdział 8. **Z przecinkiem za pan brat ... 175**
Serie danych CSV bez nagłówka ... 176
Serie danych CSV z nagłówkami ... 178

Rozdział 9. **Czas na Pythona ... 181**
Moduł time ... 182
Moduł calendar ... 184
Moduł datetime ... 185

Rozdział 10. **Zobaczyć i uwierzyć ... 189**
Instalacja biblioteki Matplotlib ... 189
Kłopotliwy Windows ... 190
Pierwszy wykres ... 191
Modyfikacje wyglądu wykresu ... 192
Wykresy wielokrotne ... 194
Prosta analiza danych ... 195
Wykresy słupkowe ... 196
Histogramy ... 197
Integracja z danymi CSV ... 198
Podręcznik Matplotlib na bezludną wyspę? ... 199

Rozdział 11. **Kłopotliwe okienka ... 201**
Instalacja ... 202
Pierwsze okienka... ... 202
Przegląd możliwości Easy GUI ... 203
Okno komunikatu (msgbox) ... 204
Okno kontynuacji (ccbox/ynbox) ... 204
Okno wyboru (buttonbox) ... 205
Lista wyboru (choicebox) ... 205
Formularze wprowadzania danych (multenterbox) ... 206
Selektor wyboru pliku lub katalogu z dysku ... 207
Miniedytor lub panel podglądu tekstu (codebox) ... 208
Podsumowanie ... 209

Rozdział 12. **Szybkie tablice NumPy ... 211**
Instalacja ... 212
N-wymiarowe tablice NumPy ... 212
Tablice i macierze NumPy ... 212

Deklarowanie tablic i macierzy NumPy ... 213
Funkcje tablicowe NumPy ... 215
Zmiany układu i rozmiaru tablic NumPy ... 218
Wycinki w tablicach NumPy ... 220
Użycie struktur NumPy w Matplotlib ... 221
NumPy — podsumowanie ... 225

Rozdział 13. **Nakarmić Pandas danymi! ... 227**
Czego potrzebujemy ... 228
Model danych w bibliotece Pandas ... 228
Obiekty Pandas Series ... 229
Obiekty Pandas DataFrame ... 231
Import danych zewnętrznych ... 233
Czyszczenie danych ... 236
Analiza jadłospisu pandy ... 240
Wizualizacja z użyciem Matplotlib ... 242

Rozdział 14. **Python i Excel ... 243**
Czego potrzebujemy ... 243
Otwieramy pliki Excela ... 244
Otwieranie skoroszytów i arkuszy danych ... 244
Odczytywanie zakresów danych ... 246
Zapis danych do skoroszytu Excela ... 247
Modyfikacja struktury ... 248

Rozdział 15. **Przeszukiwanie logów ... 251**
Analiza tre
ści plików z linii poleceń ... 251
System Windows ... 252
Systemy Linux ... 253
Wyrażenia regularne ... 255
Realizacja regex w Pythonie ... 256

Podsumowanie ... 259

Skorowidz ... 261

Przedmowa

Kurtyna w górę, na scenę wychodzi Autor.

— Tak, to prawda… Mam w ręku moją nową książkę *Python dla testera*.

Publiczność milczy skonsternowana.

— Jest to podręcznik Pythona! — zachęca Autor.

Skrzypnęło kilka siedzeń, ale nikt się nie odezwał.

— Kolejny? — Odezwał się w końcu jakiś głos z sali.

*

Ta wyimaginowana scenka ilustruje efekt, którego chcę oczywiście uniknąć. Na rynku wydawniczym można spotkać sporo podręczników języków programowania i ciężko się na pierwszy rzut oka zorientować, czym się one różnią[1]. Nie chcąc powielać konwencji podręczników pisanych przez informatyków dla informatyków, chciałbym zaproponować podręcznik Pythona w nieco innej formule, publikację tematyczną skierowaną do **testerów oprogramowania**. Dlaczego *testerów* wyjaśnię w dalszej części wstępu, gdzie skupię się także na genezie książki.

Otóż pisząc ten podręcznik, chciałem zilustrować drogę, którą osobiście przeszedłem jako menedżer zespołu testerów oprogramowania — wyzwanie polegające na poznaniu języka, który jest używany przez inżynierów niemal codziennie do **automatyzacji zadań** związanych z testowaniem. Zadanie powiodło się i dzięki poznaniu Pythona uzyskałem skuteczne narzędzie pozwalające na techniczne dyskusje z moim zespołem.

To, że moje doświadczenie wynika z pracy w dużej, profesjonalnej firmie, misyjnie wręcz zaangażowanej w procesy jakościowe, nie ma znaczenia — także w mniejszych firmach oferujących oprogramowanie widać wzrastające zainteresowanie Pythonem, który umożliwia niesamowity skok wydajności w porównaniu z klasycznym testowaniem polegającym na klikaniu, wpisywaniu komend, ręcznej modyfikacji parametrów brzegowych testów.

Testy automatyczne realizowane w firmie, w której pracuję[2], są oparte na platformie Robot Framework. Tak zwane testy manualne także są wspomagane przez skrypty pisane w Pythonie — w każdym przypadku znajomość tego języka jest kluczowa dla osób wykonujących pracę testera w profesjonalnym środowisku opartym na tzw. ciągłej integracji.

---

[1] Jakby się jednak zastanowić… z tego, co mi wiadomo, żaden podręcznik Pythona nie zachęca do nauki cytatami z wierszy Emily Dickinson!

[2] Opis dotyczyć może tak naprawdę każdej innej firmy zajmującej się wytwarzaniem oprogramowania na dużą skalę, gdy praca wielu zespołów scrumowych kończy się dostawami w środowisku ciągłej integracji. W tej branży praktyki są bardzo podobne do siebie.

> Przypomnę, że koncepcja ciągłej integracji (ang. *continuous integration*, CI) polega na stałym dostarczaniu zmian w oprogramowaniu, co wymusza uruchamianie testów wstępnych (ang. *smoke tests*) oraz testów akceptacyjnych (ang. *acceptance tests*). Środowiska CI są często budowane na serwerach Jenkinsa oraz wzbogacane rozmaitymi systemami realizującymi testy bez udziału człowieka. Systemy CI oraz testy automatyczne oczywiście nie pokrywają całości potrzeb procesów kontroli jakości, do której wymagane są dodatkowo testy obciążeniowe i dogłębne testy regresyjne.

Pomijając specyfikę związaną z tematyką testowania, chciałem poznać Pythona w stopniu wystarczającym np. do skutecznego rozwiązywania drobnych problemów dotyczących narzędzi używanych w moim osobistym środowisku pracy, np. do efektywniejszego przetwarzania pewnych danych operacyjnych używanych przez firmę, w której pracuję (przetwarzania raportów z zewnętrznych aplikacji, np. wyciągów z systemów HR, raportów z systemów zarządzania alokacjami ludzi do projektów).

Pisząc tę książkę, cały czas zastanawiałem się, jak dopasować zakres tematyczny i poziom skomplikowania materiału, tak **aby zachęcić, a nie zniechęcić** do nauki Pythona osoby dopiero raczkujące w branży IT (a często są to ludzie, którzy nigdy nie uzyskali wykształcenia *stricte* informatycznego). Było to pewnym wyzwaniem, ale od razu przyznam uczciwie, że miałem nieco ułatwioną sytuację, gdyż dla mnie Python był po prostu kolejnym językiem programowania, który napotkałem na ścieżce kariery w branży ogólnie nazywanej IT, a ponadto z uwagi na moje doświadczenie (trener, autor książek informatycznych) jestem szczególnie wyczulony na techniki skutecznego przekazywania wiedzy. Jestem też przekonany, że tzw. bariera wejścia w przypadku nauki Pythona jest znacznie niższa w porównaniu z wyzwaniami czekającymi na adeptów Javy lub C++.

Z tych obserwacji i doświadczeń narodził się mój **pomysł na podręcznik Pythona przeznaczony dla testerów**:

- Zapoznasz się z takim zakresem składniowym i filozofią tego języka, aby jak najszybciej móc zacząć go efektywnie używać do rozwiązywania zagadnień czysto praktycznych.
- Od raz na początku umawiamy się, że nie musisz wcale stać się programistą Pythona, zanurzonym w obszernym uniwersum i zafascynowanym rozpoznawaniem jego niuansów[3]. Czasem mniej oznacza lepiej, a przecież chcę Cię zachęcić, a nie przestraszyć!
- Jest zauważalne, że podręczniki języka lub kursy internetowe Pythona często zawierają koszmarnie nieużyteczne przykłady kodu… Zapewniam, że w tej książce odnajdziesz wartościowe konkrety i gotowe rozwiązania, które jakoś umykają innym autorom.
- A co, jeśli zachęcony lekturą tej książki zechcesz dowiedzieć się, gdzie szukać wiedzy? Wtedy już oczywiście możesz inwestować w rozmaite kompendia, podręczniki zajmujące ponad 1000 stron… (są takie!) — jednak dzięki lekturze niniejszego podręcznika na pewno wybierzesz kolejną książkę w sposób bardziej świadomy!

Czy to jest książka wyłącznie dla testerów? Oczywiście nie, przecież casus testera-nie-informatyka nie jest jedynym, kiedy osoba na co dzień nieprogramująca poznaje jakąś technologię, aby ją stosować wyrywkowo lub w określonym zakresie — takimi przykładami mogą być w Pythonie np. obszary nauki o danych (za tym hasłem kryje się zaawansowana analityka danych) lub uczenia maszynowego, czyli domen wcale nie zarezerwowanych dla informatyków, ale dla zwykłych użytkowników komputerów.

[3] Tak, każdy język programowania ma swoje pułapki, w które prędzej czy później wpadniesz — prostota Pythona często prowadzi do zaskakujących, ciężkich do wykrycia błędów.

Podsumowując:

Wykonując zawód testera oprogramowania (to jest motyw przewodni tej książki), możesz niesamowicie zwiększyć swoją wydajność przez stworzenie w Pythonie zbioru narzędzi pomagających Ci w pracy!

Jeśli szukasz możliwości zatrudnienia w branży IT/ICT, to przestudiowanie tego podręcznika może Ci otworzyć drzwi do niejednego pracodawcy (a piszę to w sposób bardzo odpowiedzialny, gdyż w ramach swoich obowiązków zawodowych aktywnie zajmuję się procesami rekrutacyjnymi).

Niezależnie jednak od celu, jaki Ci przyświeca, sam język musisz najpierw skutecznie poznać, w czym postaram się… także skutecznie (mam nadzieję) Ci pomóc!

Pisząc tę książkę, zastosowałem podejście, które sprawdziło się w kilku innych moich poprzednio wydanych książkach:

- Postaram się przede wszystkich nie zanudzić.
- Znajdziesz tutaj minimum teorii i maksimum praktyki.
- Przetestujemy przykładowy, gotowy do użycia kod, który może posłużyć jako punkt wyjścia do rozwijania większych aplikacji.

Mam zatem nadzieję, że ten tytuł, podobnie jak inne moje książki, także wzbudzi zainteresowanie czytelników!

Dlaczego Python?

Pythonem szerzej zainteresowałem się, gdy dostrzegłem wzrastającą popularność tego języka wśród administratorów, testerów i programistów używających go jako prostego środowiska do automatyzacji pracy, a czasami nawet do tworzenia zaawansowanych aplikacji biznesowych i telekomunikacyjnych.

Język ten jest często kojarzony z szeroko pojętym uczeniem maszynowym (ang. *Machine Learning*) i sztuczną inteligencją (ang. *Artificial Intelligence*)[4] z powodu doskonałych bibliotek ML i AI dostępnych za darmo i używających właśnie Pythona. Sam język nie jest już nowinką; jego twórca Guido van Rossum opracował pierwszą wersję na początku lat 90. i to, że do dziś jest tak intensywnie rozwijany, posiada dynamiczną społeczność internetową i zyskuje popularność nawet wśród osób nietechnicznych, gwarantuje, że inwestycja w jego poznanie będzie opłacalna!

Pozycję Pythona utwierdzają też rankingi branżowe, np. firma analityczna RedMonk po raz kolejny potwierdziła dominującą pozycję tego języka. W raporcie ze stycznia 2021 r. (*https://redmonk.com/sograd y/2021/03/01/language-rankings-1-21*) umieszczono go na drugim miejscu, wyżej niż Javę!

1. JavaScript.
2. Python — trzy lata wcześniej znajdował się na pozycji trzeciej i stale pnie się w górę zestawienia!
3. Java.
4. (i dalej): PHP, C#, C++…

[4] Według mnie nawet zbyt często, co powoduje szereg nieporozumień dotyczących jego możliwych zastosowań.

Analitycy RedMonka sukcesu Pythona upatrują w łatwości użycia (prosta i czytelna składnia, bez uciążliwej i znanej z innych języków „ornamentowej" otoczki klas wywoławczych, zasad komponowania funkcji *main* itp.) i niskiej barierze wejścia (szybciej się nauczysz Pythona niż niuansów Javy lub C++). Kolejnym argumentem jest nieprawdopodobna liczba domen biznesowych i technicznych, które ten język opanował: zastosujesz go zarówno w prostych aplikacjach technicznych, jak i w zaawansowanej analizie danych, bez problemu możesz go użyć do pisania aplikacji biznesowych.

Python jest językiem skryptowym i podobnie jak bardzo niegdyś popularny Perl może być uruchamiany „z palca" użytkownika (po prostu wpisujesz komendy) lub poprzez załadowanie gotowego pliku z kodem, który może być utworzony w najprostszym edytorze tekstowym. Co więcej, jeśli planujesz ukryć zawartość skryptów, to można skompilować program do wersji wykonywalnej (np. EXE w Windowsie) i dystrybuować już program w wersji binarnej.

Dla kogo nie jest przeznaczona ta książka?

We wstępie staram się przekonać czytelnika, że Python potrafi wiele (niektórzy wręcz sugerują, że potrafi wszystko, ale jest to semantyczne nadużycie wynikające z istnienia rozlicznych i darmowych bibliotek użytkowych). Tę książkę pisałem, niejako ilustrując drogę, którą sam przeszedłem podczas nauki Pythona, co niejako domyślnie oznacza, że nie jest to pozycja dla zawodowych programistów, którzy znając podstawy, znajdują się raczej na etapie wymagającym studiowania pozycji o charakterze kompendium lub których interesują określone biblioteki stworzone dla tego języka.

Dla kogo JEST przeznaczona ta książka?

Nieco uogólniając: książka jest przeznaczona dla osób w miarę biegłych w obsłudze komputerów, zaawansowanych użytkowników, którym brakuje narzędzi do automatyzacji zadań. Zgodnie z tytułem zawiera ona materiał mogący zainteresować zawodowych testerów, tzw. manualnych, którzy pragną zwiększyć wydajność swojej pracy poprzez użycie Pythona np. w celu automatyzacji wykonywania testów.

Kogo może ponadto zainteresować zawarty w niej materiał?

Używasz komputera do nauki lub w celach zawodowych?

Jeśli jesteś uczniem lub studentem znającym systemy operacyjne, testerem lub integratorem, administratorem, programistą pracującym w firmie wyposażonej w środowisko IT — na pewno wykonujesz wiele czynności, które możesz usprawnić lub zautomatyzować i Python nadaje się do tego celu doskonale!

Zarządzasz zespołami IT lub deweloperskimi?

Sam jestem menedżerem, zatem z pewną dozą nadziei napiszę, że być może tytuł ten zainteresuje innych menedżerów w działach IT lub firmach tworzących oprogramowanie. Kto wie, czy odrobina wiedzy na temat narzędzi używanych przez Twoich inżynierów-specjalistów nie pozwoli Ci lepiej formułować oczekiwania i definiować wymagania?

Mój podręcznik może zatem zainteresować menedżerów nadzorujących zespoły, które wykonują szereg powtarzalnych zadań — Python pomoże zwiększyć ich wydajność i jeśli poznasz możliwości tego języka, dyskusja o wymaganiach z osobami technicznymi stanie się prosta.

Interesuje Cię analiza danych lub obszary AI/ML?

Jeśli jesteś analitykiem danych lub naukowcem, to nadszedł najwyższy czas, aby poznać Pythona na poziome co najmniej podstawowym, żeby móc używać powszechnie dostępnych bibliotek w tym języku. Nie trzeba być bowiem ekspertem, aby tworzyć proste skrypty lub dopasowywać już istniejące. Przetwarzanie danych, uczenie nadzorowane lub nienadzorowane, pisanie kodu dla sieci neuronowych wymaga bowiem bardziej zrozumienia gotowych bibliotek (np. Scikit-Learn, TensoFlow) niż samego Pythona, który jest po prostu środowiskiem uruchomieniowym, używanym do zdefiniowania określonych parametrów (np. grafu obliczeniowego w TensorFlow). Czy to jest Python? Tak, ale tylko *operacyjnie*, gdyż cała magia tkwi we wspomnianych bibliotekach!

A może chcesz zmienić branżę?

Nietrudno zauważyć, że rynek pracy jest coraz bardziej nastawiony na obszary IT/ICT. Znam wiele osób, które z sukcesem przebranżowiły się, znajdując zatrudnienie jako testerzy w firmach produkujących oprogramowanie lub towary przemysłowe. Jeśli interesuje Cię taka ścieżka kariery, to poważanie zastanów się, czy nie zwiększyć swoich kompetencji, inwestując w kursy systemów operacyjnych (Windows, Linux), podstawy sieci komputerowych i… jakiś popularny język programowania.

Osobiście oceniam, że poznanie Pythona jest w granicach możliwości każdej osoby jako tako biegłej w obsłudze komputera i znającej angielski. Nieinformatykom znacznie trudniej jest rzetelnie poznać kanoniczne języki programowania (np. Javę, C++), gdyż są one obiektywnie znacznie trudniejsze i wskazane jest jednak posiadanie odpowiedniego wykształcenia kierunkowego obejmującego np. teorię algorytmów i dogłębną znajomość struktur danych.

Co dalej?

Struktura tego podręcznika jest bardzo prosta i zakłada przejście przez trzy etapy nauki:

Etap 1. Niezbędnik *narzędziowy*, czyli środowisko pracy (instalacja Pythona i podstawowych narzędzi, dokładanie bibliotek wzbogacających środowisko, praca w linii poleceń Windowsa lub Linuksa).

Etap 2. Niezbędnik *składniowy*, czyli minimum wiedzy o poprawnej strukturze programów w Pythonie (np. typy danych, wyrażenia warunkowe, podstawowe instrukcje, pętle).

Etap 3. W głównej części książki poznasz, na podstawie szeregu użytecznych programów przykładowych, możliwości Pythona i jego najbardziej popularnych bibliotek. Nauczysz się pracować z plikami, poznasz tajniki analizy i wizualizacji danych, sposoby importowania plików CSV, metody analizy logów i wiele innych, Elementy języka pominięte wcześniej będziesz poznawał nie na sucho, ale na praktycznych przykładach.

Przykładowe programy

Programy opisane w książce zostały umieszczone na serwerze FTP wydawnictwa Helion pod adresem *https://ftp.helion.pl/przyklady/pyttes.zip*. Pliki źródłowe mogą być pełniejsze i bardziej rozbudowane niż warianty prezentowane w wersji drukowanej. Można zatem założyć,

że jeśli w trakcie wykładu jest prezentowana jakaś funkcja bez podania *explicite* sposobu jej użycia, to na pewno wersja źródłowa zawiera reprezentacyjny przykład jej zastosowania. Warto zatem podczas lektury porównywać wersje umieszczone na FTP z tymi, które zostały omówione na kartach książki!

Uwaga: w celu zidentyfikowania, gdzie w książce jest opisywany plik źródłowy dostępny na serwerze FTP, możesz posłużyć się skorowidzem.

Konwencje typograficzne i oznaczenia

Poniżej znajduje się kilka typowych oznaczeń i konwencji, które zastosowano na kartach tej książki.

Regułą jest, że wszystkie listingi i teksty ukazujące się na ekranie, odróżniono od zasadniczej treści książki czcionką o `stałej szerokości znaków`; to samo dotyczy komend systemu operacyjnego, jeśli takie są opisywane.

Ważna uwaga — materiał istotny dla zrozumienia omawianego zagadnienia.

Ostrzeżenie — rzeczy, których nie powinieneś robić, jeśli chcesz uniknąć kłopotów.

Odwołanie — w miejscu, które wskazuje, znajdziesz dodatkowe informacje dotyczące omawianego zagadnienia.

Prog.py

Taki symbol oznacza, że tekst programu znajduje się w pliku *Prog.py*, umieszczonym w archiwum ZIP zawierającym przykładowe programy dostępne na serwerze FTP firmy Helion. W niektórych rozdziałach napotkasz kod wbudowany w treść akapitu bez oznaczenia nazwy pliku — zazwyczaj jest to kontynuacja opisu rozpoczętego kilka stron wcześniej; w takiej sytuacji cofnij się o kilka stron i sprawdź, czy nazwa pliku nie została już podana. Zalecam rozpakowanie archiwum ZIP w dowolnym katalogu, co automatycznie utworzy folder o nazwie *przyklady* zawierający skrypty, pliki robocze oraz podfoldery używane przez niektóre skrypty.

Komendy poleceń lub wyniki działania programu w tekstowej konsoli systemowej, np. cmd w Windowsie lub Terminal w Linuksie albo macOS-ie.

Początki są trudne, ale przy odrobinie wytrwałości możesz zadziwiająco szybko nabyć sprawność „kodowania", do tej pory zarezerwowaną dla programistów.

Zapraszam do lektury i chciałbym na koniec bardzo wyraźnie ostrzec Cię, Czytelniku, że wkraczasz w słynną „Strefę wolną od zbędnej teorii"!

Piotr Wróblewski,
Wrocław marzec – czerwiec 2021

Rozdział 10.

Zobaczyć i uwierzyć

Drogi Czytelniku, bez obawy, autor po przeżyciu kolejnego lockdownu i napisaniu dziewięciu poprzednich rozdziałów jeszcze nie oszalał... Ta książka ciągle jest po prostu podręcznikiem Pythona przeznaczonym dla testerów, a tytuł rozdziału odnosi się po prostu do **wizualizacji danych**, która często stanowi uwiecznienie żmudnej pracy wykonywanej podczas ręcznie lub automatycznie przeprowadzanych serii testów.

Do tej pory omówiłem na tyle bogaty materiał kanoniczny dotyczący samego języka, że pora przejść na nieco wyższy poziom interakcji, jaką jest możliwość skutecznego zaprezentowania informacji. Umiejętność ta jest przydatna także podczas testowania systemów. Często zdarza się, że badane są odpowiedzi systemu na określone „bodźce stymulujące", np. zmieniające się zestawy parametrów wejściowych. W rozdziale poprzednim omówiliśmy format CSV, który pozwala na skutecznie przekazywanie serii danych pomiędzy systemami lub odbieranie danych np. z urządzeń (poprzez interfejsy aplikacyjne). Takie zestawy informacji często są bardzo duże i również zachodzi potrzeba ich analizy, np. w celu wykrycia anomalii lub korelacji między seriami danych.

W celu analizy informacji, jakie skrywają w sobie „surowe" serie danych, można podejść do tematu „matematycznie", tj. wykorzystać biblioteki, które nam to ułatwią, pomagając w oczyszczaniu danych i umożliwiając wszechstronne analizy statystyczne. Niestety wkraczamy wówczas w bardzo szeroki obszar analizy danych (czasami nazywanej pompatycznie „nauką o danych", ang. *data science*), co oczywiście znacznie wykracza poza zakres tej książki. Okazuje się jednak, że nawet bez porywania się na opisy pewnych „naukowych" bibliotek Pythona można łatwo wytłumaczyć klasyczne metody tworzenia graficznych wykresów oraz ich zapisu do dokumentów PDF. Nawet takie proste realizacje często pozwalają wykryć zaburzenia, anomalie, nieoczywiste korelacje — czasem na pierwszy rzut oka. Ciężko jest jednak analizować tabele danych przedstawiane jako serie liczb i wówczas mogą nam pomóc proste wizualizacje.

O tym wszystkim opowiem w tym rozdziale i przekonasz się, że nie są to trudne zagadnienia.

Instalacja biblioteki Matplotlib

Do tej pory moje opisy zakładały używanie wyłącznie wbudowanych bibliotek Pythona i jak się przekonaliśmy, nawet w tym standardowym środowisku, przy odrobinie wyobraźni połączonej z wiedzą, da się zrobić bardzo dużo.

Teraz jednak nadszedł czas na użycie pakietu dodatkowego o nazwie Matplotlib, a ponieważ jest on utrzymywany niezależnie, musisz go wgrać komendą `pip`. Moduł ten oferuje bibliotekę klas pozwalających na tworzenie wizualizacji w Pythonie (statycznych, dynamicznych, a nawet interaktywnych), np. wykresów funkcji.

Poniżej, w terminalu linii poleceń, pokazałem fragment sesji, który ilustruje proces instalacji lub aktualizacji tej biblioteki w moim systemie (wymagane jest połączenie z Internetem, w przypadku systemu Windows użyj komendy `python` zamiast `python3`).

Terminal

```
python3 -m pip install -U pip
python3 -m pip install -U matplotlib
Collecting matplotlib
  Downloading matplotlib-3.4.2-cp39-cp39-macosx_10_9_x86_64.whl (7.2 MB)
...
Successfully installed cycler-0.10.0 kiwisolver-1.3.1 matplotlib-3.4.1 numpy-1.20.2
↪pillow-8.2.0 pyparsing-2.4.7 python-dateutil-2.8.1 six-1.16.0
Installing collected packages: matplotlib
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.4.1
    Uninstalling matplotlib-3.4.1:
      Successfully uninstalled matplotlib-3.4.1
Successfully installed matplotlib-3.4.2
$ pip show matplotlib
Name: matplotlib
Version: 3.4.1
```

Jeśli ujrzysz podobne komunikaty, to oznacza, że instalacja biblioteki powiodła się. Jak łatwo zauważyć, wraz z Matplotlib instalowane są dodatkowe moduły, których on wymaga, np. NumPy, specjalistyczna biblioteka ułatwiająca obsługę dużych, wielowymiarowych tablic i macierzy, które mogą stanowić źródła danych dla wykresów tworzonych z użyciem Matplotlib.

Autorami biblioteki są John D. Hunter i Michael Droettboom. Gdy już zapoznasz się z omówieniem zamieszczonym w tym rozdziale, koniecznie odwiedź stronę zawierającą źródłowe informacje referencyjne opisujące wszystkie metody zawarte w tej bibliotece: *https://matplotlib.org*.

Kłopotliwy Windows

Jeśli używasz systemu Windows i podczas testowania przykładowych programów uruchomienie nie powiedzie się, np. otrzymasz komunikat podobny do `from matplotlib._path import (ImportError: DLL load failed while importing _path: Nie można odnaleźć określonego modułu`, to zainstaluj dodatek o nazwie *Microsoft Visual C++ Redistributable*. Zawiera on biblioteki, które są wymagane do uruchomienia aplikacji w języku C++ skompilowanych przy użyciu programu Visual Studio 2015. Dodatek ten jest dostępny pod adresem:

https://www.microsoft.com/pl-PL/download/details.aspx?id=48145.

Pierwszy wykres

Nawet bez wczytywania się w obszerną dokumentację („obszerną" to eufemizm, sam podręcznik użytkownika zajmuje 3487 stron w wersji PDF!)[1] można szybko i bez wysiłku utworzyć prosty program tworzący wykresy.

Załóżmy, że interesuje nas stworzenie wykresu funkcji, dla której znamy dziewięć wartości (pary *x*, *y*), które w kodzie Pythona są umieszczone w listach o nazwach `osX` i `osY`.

Skrypt, który tworzy taki wykres, jest wyjątkowo zwięzły:

Listing

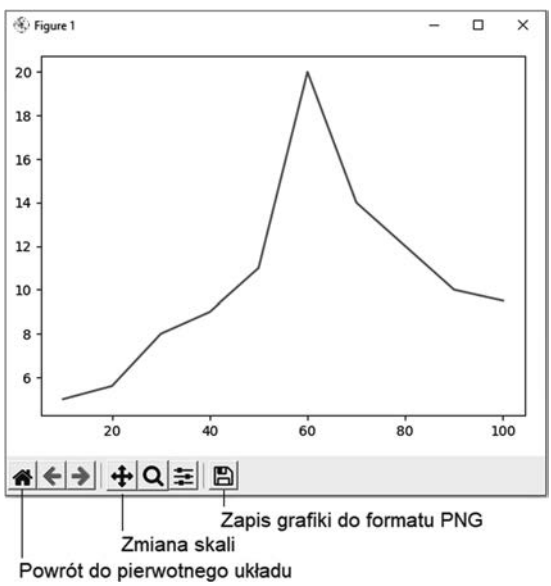
wykres0.py

```
from matplotlib import pyplot as plt
osX =[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
osY =[5, 5.6,  8,  9, 11, 20, 14, 12, 10, 9.5]
plt.plot(osX, osY)
plt.savefig('wykresik.pdf')
plt.show()
```

Pomimo że skrypt zajmuje raptem kilka linijek, to wykonuje sporo pracy:

- Wyświetla miniprogram graficzny, zawierający wykres i kontrolki pozwalające nim operować (skalowanie, nawigacja w zakresie osi *X*, *Y*).
- W ramach prezentu pożegnalnego na dysku, w katalogu roboczym zostanie zapisany plik o nazwie *wykresik.pdf*, będący kopią wykresu w jego układzie pierwotnym.

Rysunek 10.1 pokazuje okienko, które zobaczysz na ekranie po uruchomieniu skryptu (wersja Windows, w macOS-ie lub Linuksie styl okna będzie nieco inny).



Rysunek 10.1. *Wykres wygenerowany przez moduł pyplot (Matplotlib)*

[1] To nie jest żart. Tyle właśnie stron zajmuje podręcznik PDF dla wersji 3.4.2.

Kluczowymi przyciskami nawigacyjnymi są:

- przycisk  — powrót do układu pierwotnego (reset);
- przycisk  — skalowanie lub przesunięcie.

Zwróć uwagę na kilka przycisków nawigacyjnych — ich przeznaczenie jest dość intuicyjne. Przycisk resetu (symbol domku) pozwala powrócić do pierwotnego układu wykresu przed ewentualnymi zmianami skali lub przesunięciami, jakie użytkownik wykonał za pomocą myszy po wciśnięciu przycisku pozwalającego na skalowanie wykresu (zoom) lub zmianę widocznego obszaru w ramach osi *X*, *Y*.

Wspierane funkcje skalowania lub przesunięć:

- Możesz chwytać wykres myszą i przesuwać w lewo i w prawo, np. aby przejść poniżej osi *X*, w stronę ujemnych wartości *Y*.
- Wciśnij klawisz *Ctrl* i przytrzymując wciśnięty *prawy przycisk* myszy, przesuwaj ją w górę (powiększanie) lub w dół (pomniejszanie).
- Wciśnij klawisz *Ctrl* i przytrzymując wciśnięty *lewy przycisk* myszy, przesuwaj ją w górę i w dół (przesuwanie na osi *X*) lub w lewo i w prawo (przesuwanie na osi *Y*).

Ponowne wciśnięcie przycisku pozwala powrócić do klasycznego ekranu i zachowanego stanu wykresu.

Modyfikacje wyglądu wykresu

Prosty wykres zbudowany z domyślnymi ustawieniami rzadko spełnia nasze oczekiwania i może okazać się nieczytelny. Modyfikacje wyglądu wykresu wprowadzamy jako dodatkowe parametry metody wykreślającej wykres (`plot`), która w przypadku wykresów dwuwymiarowych pozwala na łatwe zmodyfikowanie koloru, grubości oraz stylu linii, można też zmienić sposób wyświetlania punktów danych.

W tym celu w funkcję `plot` wzbogacamy o parametry:

- `marker` — styl punktu danych;
- `ms` — rozmiar markera (ang. *marker size*);
- `linestyle` — styl linii;
- `color` — kolor;
- `linewidth` — grubość linii.

Tabela 10.1 zawiera listę kolorów dostępnych w ramach palety podstawowej RGB. Można podawać kolory według notacji kodowej RGB w konwencji `'#rrggbb'`, (np. `'#67e210'` pozwoli uzyskać piękny, wiosennie rozjaśniony odcień zieleni).

Tabela 10.1. *Kody kolorów palety podstawowej rozróżniane przez funkcję wykreślającą plot*

Cecha	Kolor	Cecha	Kolor
`'m'`	Purpurowy (ang. *purple*), magenta	`'r'`	Czerwony (ang. *red*).
`'y'`	Żółty (ang. *yellow*).	`'g'`	Zielony (ang. *green*).
`'k'`	Czarny (ang. *black*).	`'b'`	Niebieski (ang. *blue*)
`'w'`	Biały (ang. *white*)	`'c'`	Niebieskozielony (ang. *cyan*)

Tabela 10.2 zawiera listę kodów pozwalających zmodyfikować styl kreskowania, co jest bardzo przydatne podczas wykreślania wykresów wielokrotnych i ułatwia rozróżnianie, która linia odpowiada określonej serii danych.

Tabela 10.2. *Typy kreskowania w Matplotlib*

Kod	Styl
`'-'`	___ (linia ciągła)
`':'`	..... (linia kropkowana)
`'--'`	– – – (linia kreskowana)
`'-.'`	– . – . (linia kropkowano-kreskowana)

Tabela 10.3 zawiera kilka wybranych kodów, których można użyć do oznaczenia punktu danych tworzącego wykres.

Tabela 10.3. *Wybrane style markerów*

Kod	Styl	Kod	Styl
`'o'`	Kółko	`'P'`	Duży pogrubiony plus
`'s'`	Kwadracik	`'s'`	Gwiazdka
`'*'`	Gwiazdka	`'D'`	Diament równomierny
`'x'`	Symbol X	`'1'`	Drzewko
`'X'`	Symbol X wypełniony	`'p'`	Pięciobok
`'+'`	Plus	`'H'`	Sześciobok
`'v'`	Trójkącik w dół	`'^'`	Trójkącik w górę
`'<'`	Trójkącik w lewo	`'>'`	Trójkącik w prawo

Popatrz na przykładowe użycie wybranej kombinacji tych parametrów:

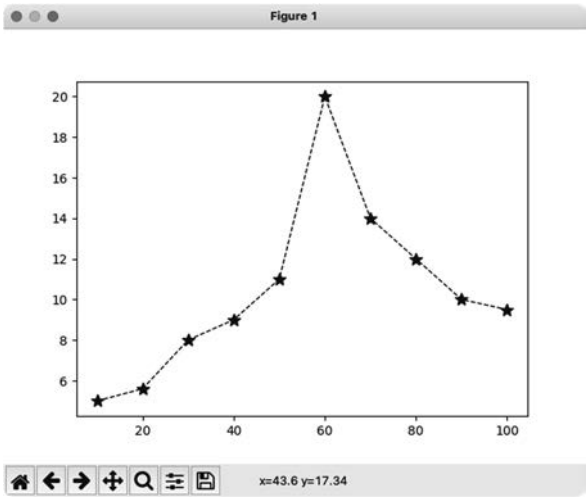
Listing

wykres1.py

```
from matplotlib import pyplot as plt
osX=[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
osY=[5, 5.6, 8, 9, 11, 20, 14, 12, 10, 9.5]
plt.plot(osX, osY,
         marker='*',          # Styl markera (tu: symbol gwiazdki)
         linestyle='--',      # Styl linii (tu: kreskowana)
         color='k',           # Kod koloru (tu: czarny)
         ms=10,               # Rozmiar markera
         linewidth = '1')     # Grubość linii
plt.show()
```

Rysunek 10.2 pokazuje wynik na ekranie.

Rysunek 10.2.
Style linii i markery



Wykresy wielokrotne

Wiele wykresów komponuje się w celu dokonania porównań kilku serii danych, co oczywiście jest wspierane przez Matplotlib. Możliwość dołożenia nowego wykresu na poprzedni polega na *kolejnym wywołaniu funkcji kreślącej* `plot()`. Oczywiście warto w tym momencie dodać kilka kolejnych ozdobników, których użycie pokażę na prostym przykładzie:

- własny tytuł wykresu,
- etykiety osi,
- legenda,
- znormalizowanie wartości na osi *X*.

A zatem do dzieła!

Listing

wykres2.py

```
from matplotlib import pyplot as plt
osX=[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
osY =[5, 5.6, 8, 9,  11, 20, 14, 12, 10, 9.5]
osY2=[2, 3,   6, 9,  12, 13, 16, 14, 12, 12 ]

seria1,=plt.plot(osX, osY,  marker='*', linestyle='--', color='k',
                 ms=10, linewidth = '1', label='Seria 1.')        # Seria 1.
seria2,=plt.plot(osX, osY2, marker='o', linestyle='-',  color='k',
                 ms=10, linewidth = '1', label='Seria 2.')        # Seria 2.

plt.legend(handles=[seria1, seria2])
plt.title("Pomiary napięcia")
plt.xlabel("[t]")
plt.ylabel("[V]")
plt.xticks([0, 25,50,75, 100])                    # Normalizacja wartości na osi X (*)
plt.show()
```

Dość specyficzna składnia użyta powyżej wynika z konstrukcji klasy Matplotlib i wymagań wobec określonych typów danych używanych do przekazywania wartości koniecznych do zbudowania tekstu legendy.

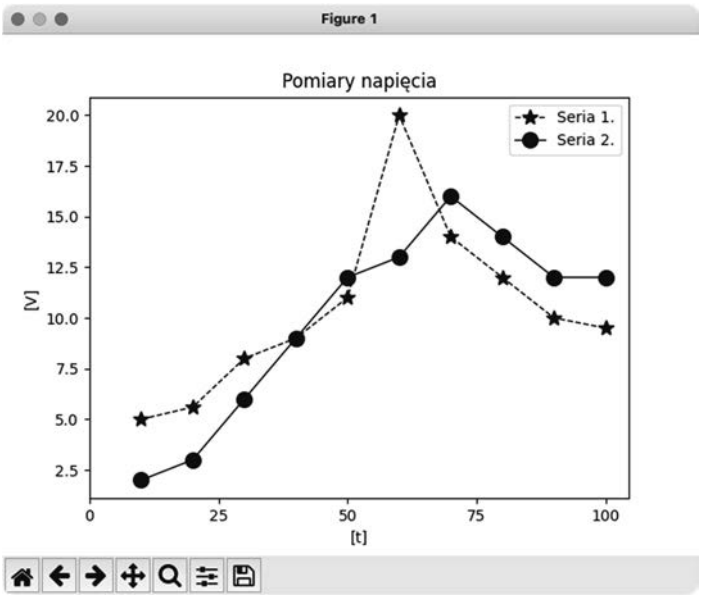
Wiersz oznaczony (*) zastępuje 10 wartości występujących na osi *X*:

```
osX= [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

ich mniejszą liczbą, na dodatek są one rozłożone równomiernie (0, 25, 50,75, 100), co czyni wykres bardziej czytelny. Ta sztuczka może też się przydać do opisowej zmiany jednostek. Na przykład gdyby wartości osi *X* były znacznikami daty i czasu, to zamiast nich można dać czytelne etykiety zakresu, np. nazwy dni tygodnia, kwartały.

Efekt końcowy jest bardzo czytelny (rysunek 10.3):

Rysunek 10.3.
Wykresy wielokrotne



Prosta analiza danych

Matplotlib oferuje oczywiście nie tylko wykresy liniowe. W praktyce testera, osoby mimo wszystko pracującej często na styku analizy danych[2], zachodzi nierzadko potrzeba przeprowadzenia nieskomplikowanych analiz rozkładu danych, np.:

- chcemy podsumować dane zebrane podczas przetwarzania innych zbiorów informacji, co skutkuje np. wykonaniem sum cząstkowych wyników zebranych w danej kategorii;
- budujemy proste wizualizacje rozkładów statystycznych.

Zapoznajmy się zatem ze sposobami tworzenia wykresów słupkowych (ang. *bar char*) i histogramów, które doskonale nadają się do takich celów.

[2] Nie używam określenia „nauka o danych”, tj. *data science*, bowiem byłoby to zdecydowanym nadużyciem!

Wykresy słupkowe

W tym punkcie omówimy sposób prezentowania danych w formie wykresu słupkowego.

Analizując ten przykład, nauczysz się następujących elementów:

- używania innych serii danych niż liczbowe,
- definiowania wykresu słupkowego,
- normalizowania wartości na osi *Y*.

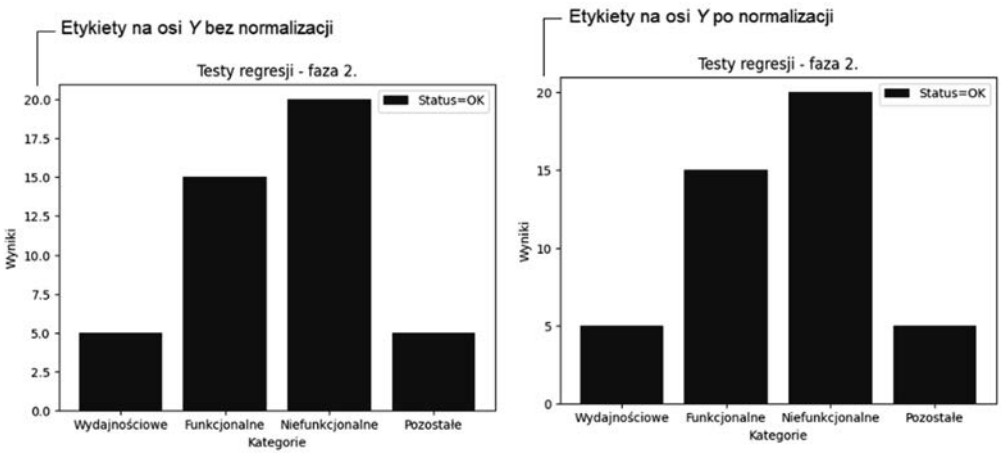
Treść naszego wykresu może reprezentować statystyczny rozkład pewnych informacji — niech to będzie np. liczba pozytywnie zakończonych testów regresji, z podziałem na kategorie. Jedna z możliwych realizacji jest pokazana poniżej:

Listing

wykres3.py

```
from matplotlib import pyplot as plt
testy=['Wydajnościowe', 'Funkcjonalne', 'Niefunkcjonalne', 'Pozostałe']
passRate =[5, 15, 20, 5]
statystyki=plt.bar(testy, passRate,  color='k', label='Status=OK')
plt.legend(handles=[statystyki])
plt.title("Testy regresji - faza 2.")
plt.xlabel("Kategorie")
plt.ylabel("Wyniki")
plt.yticks([0, 5,10,15,20]) #(*)
plt.show()
```

Efekt końcowy jest bardzo czytelny (rysunek 10.4).



Rysunek 10.4. *Wykres słupkowy*

Jak łatwo zauważyć, rolę umownej osi *X* pełni tu zbiór etykiet, czyli nazwy kategorii testów, a wyniki na osi *Y* są wyśrodkowane względem tych etykiet.

Jeśli uruchomisz program bez wiersza oznaczonego gwiazdką, to na osi pionowej *Wyniki* znajdą się liczby z zakresu podanego w tabeli `passRate`, ale efekt okaże się dziwaczny: z czterech liczb moduł zrobi rozpiskę zawierającą takie liczby jak 2.5, 7.5 itp. (lewa strona rysunku).

Tymczasem w tym konkretnym przypadku użycia takie etykiety okazałyby się mocno mylące, gdyż z natury swojej liczba wyników może być tylko liczbą całkowitą! Stąd zastosowana normalizacja etykiet na tej osi (prawa strona rysunku).

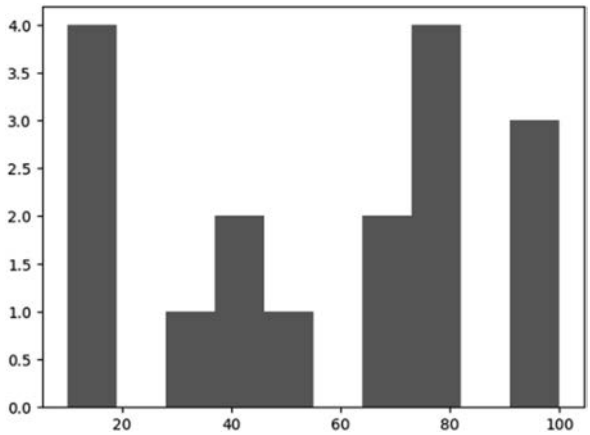
Histogramy

Histogramy zbliżają nas już ku analizie statystycznej, pomagającej zrozumieć dystrybucję (rozkład) danych według ich wartości. O ile w przypadku małych zbiorów danych łatwo jest zrozumieć, patrząc na surowe dane, które wartości są bardziej popularne, a które występują rzadziej, to już w przypadku tabeli z np. tysiącem wartości ocena w trybie „na oko" mogłaby prowadzić do błędów.

Popatrzmy zatem, jak tworzy się wykres typu histogram przy użyciu biblioteki Matplotlib[3]. Załóżmy, że mamy zebranych 17 wyników dość podobnych do siebie i chcemy utworzyć histogram, który je przenalizuje i pokaże, które z nich występują częściej, a które rzadziej — niech wykres rozrzuci te podsumowania na 10 umownych „kubełków".

Efekt końcowy pokazuje rysunek 10.5.

Rysunek 10.5. *Histogram*



Jak łatwo zauważyć, rolę umownej osi *X* pełni tu z góry ustalona liczba „kubełków" zawierających podsumowania wyników, a na osi *Y* możemy odczytać ich liczbę.

Listing

wykres4.py

```
from matplotlib import pyplot as plt
t=[10, 10, 10, 10, 30, 40, 40, 50, 70, 70, 80, 80, 80, 80, 100, 100, 100]
statystyki=plt.hist(t, 10)
plt.show()
```

[3] Jeśli masz kłopot ze zrozumieniem zasady konstruowania histogramów, to zajrzyj też do przykładu omówionego w rozdziale 12.

Integracja z danymi CSV

Po lekturze rozdziału 8. z pewnością rozumiesz, jak użytecznym narzędziem jest wymiana informacji w formacie CSV. Pokazane tam metody zapisu i odczytu mają charakter uniwersalny i oczywiście można je zastosować do pobierania danych, które chcielibyśmy poddać wizualizacji za pomocą biblioteki Matplotlib.

Tylko… po co? Okazuje się, że zamiast żmudnego parsowania wierszy z pliku CSV można użyć modułu NumPy, który zawiera gotową, jednolinijkową metodę pozwalającą na załadowanie macierzy wartościami pobranymi z pliku. Bez wnikania w bogate możliwości biblioteki NumPy, o której opowiem szerzej w rozdziale 12., skorzystamy teraz z tego mechanizmu, udowadniając raz jeszcze, że leniwi ludzie to nie nieroby, tylko inteligentne osoby, które chcą swoje zadania wykonywać sprawniej, aby w zaoszczędzonym czasie zrobić coś innego (lub się poobijać, co jest wskazane od czasu do czasu).

Na użytek ćwiczenia utworzyłem plik *danedowykresu.csv*, który zawiera te same dane, jakie wykorzystano w ćwiczeniu *wykres2.py* z tego rozdziału:

Wersja z *wykres2.py*

```
osX=[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
osY =[5, 5.6, 8, 9,  11, 20, 14, 12, 10, 9.5]
osY2=[2, 3,   6, 9,  12, 13, 16, 14, 12, 12 ]
```

Plik *danedowykresu.csv*

```
10,     5,      3
20,     5.6,    3
30,     8,      6
40,     9,      9
50,     11,     12
60,     20,     13
70,     14,     16
80,     12,     14
90,     10,     12
100,    9.5,    12
```

Co ciekawe, nowy skrypt jest niemal identyczny!

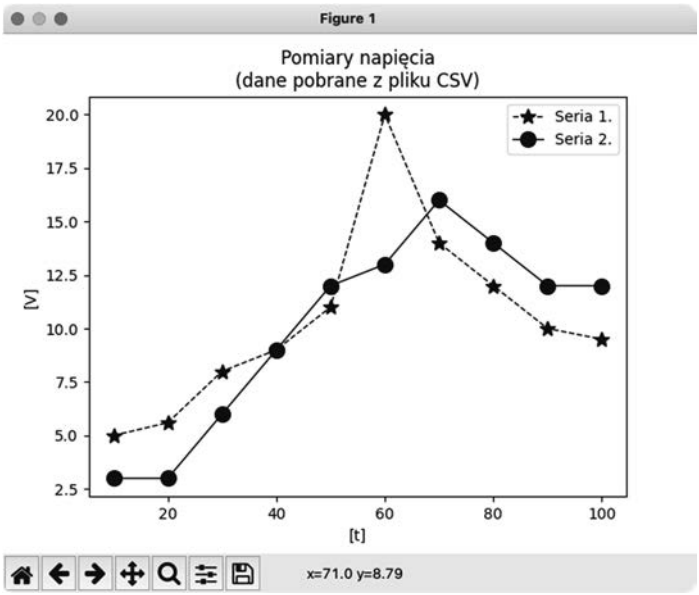
Listing

wykrescsv.py

```
import matplotlib.pyplot as plt
import numpy as np

osX, osY, osY2 = np.loadtxt('danedowykresu.csv',
                                    delimiter=',', unpack=True)
seria1,=plt.plot(osX, osY,  marker='*', linestyle='--', color='k',
                 ms=10, linewidth = '1', label='Seria 1.') # Seria 1.
seria2,=plt.plot(osX, osY2, marker='o', linestyle='-',  color='k',
                 ms=10, linewidth = '1', label='Seria 2.') # Seria 2.
plt.title("Pomiary napięcia\n(dane pobrane z pliku CSV)")
plt.legend(handles=[seria1, seria2])
plt.xlabel("[t]")
plt.ylabel("[V]")
plt.legend()
plt.show()
```

Można się bardzo łatwo przekonać, że generowany wykres jest identyczny, zmieniłem tylko jego tytuł, aby udowodnić czytelnikowi, że nie ma tu żadnego oszustwa (rysunek 10.6).



Rysunek 10.6. *Wykresy wielokrotne — dane pobrane z pliku CSV*

Podręcznik Matplotlib na bezludną wyspę?

Tak jak pisałem, dokumentacja Matplotlib jest olbrzymia i nie sposób ją powielić w sposób racjonalny. W związku z tym, aby wytłumaczyć filozofię używania tej biblioteki, postanowiłem zaznajomić czytelnika z kilkoma schematami tworzenia i upiększania wykresów.

Oczywiście nie musisz jej drukować w celu zabrania na bezludną wyspę, jeśli takie masz plany życiowe. Wystarczy, że wyspa będzie w zasięgu Internetu, wówczas wystarczy zapisać w przeglądarce zakładkę *https://matplotlib.org* i sięgać tam w razie potrzeby.

Do czego gorąco zachęcam, bo efekty mogą okazać się spektakularne!

Skorowidz

", 39
.fieldnames, 178
.shape, 218
    @property, 85
__add__, 89
__init__, 81, 86
__init__.py, 113
__mul__, 89
__pow__, 89
__sub__, 89
__truediv__, 89
init.py, 114
| (symbol pipe), 44
1D, 212
2D, 212

A

add(), 135, 217
Anakonda, 15
analiza danych, 227
analizaHTTP.py, 137
analizaWWW2.py, 141
apostrof pojedynczy, zwykły, 119
append(), 125
apply(), 239
arange(), 215
argparse, 99
ArithmeticError, 107
arkusz, 244
Artificial Intelligence, *Patrz* Sztuczna Inteligencja
ASCII, 149
asciitime(), 184
asin(), 60
atan(), 60

B

backslash, 39
bajt, 50
bash.sh, 111
Bill Gates, 259
bin (katalog), 40
bin(), 50
binarne.py, 156
binarne1.bin, 158
binarne1.py, 159
binarne2.py, 159
bool, 239
boot (katalog), 40
buttonbox(), 205

C

calculate_dimension(), 247
calendar (moduł), 184
cat, 43, 253, 254
ccbox, 204
ccbox(), 204
cd, 40
ceil(), 60
center(), 119
chmod, 43
choicebox(), 205
chown, 43
CI, *Patrz* continuous integration
clear, 41
clear(), 125, 135, 141
CLI, *Patrz* command line interface
close(), 149
cls, 41
codebox, 209
codebox(), 209
color, 192

command line interface,
  Patrz wiersz linii poleceń
compile(), 258
complex.py, 89, 100
complex128, 215
complex64, 215
continuous integration,
copy, 42
cos(), 60
count(), 120, 125, 131, 240
cp, 42
create_sheet(), 249
CSV, 175, 198, 211, 233
csv (moduł), 176
cudzysłów
    prosty, 119
    w kodzie, 48
Cyceron, 117
czasdatetime.py, 185
czasdatetime2.py, 187
czasodczyty.py, 183
czasopomiary.py, 182
czytajCSHV.py, 178
czytajCSV.py, 176
czytamypliki.py, 151
czytamyplikiPath.py, 173

D

DataFrame, 228
    zapis do Excela, 249
DataFrames, 231
datetime (moduł), 185
datetime64, 230
debugger, 23
degress(), 60
dekorator, 85
del, 42, 141
delete_cols(), 249
delete_rows(), 249
delimiter, 176
dev (katalog), 40
DictReader(), 178
diff (polecenie), 44
difference(), 136
dir, 41, 126
diropenbox(), 207
divide(), 217
długie linijki kodu, 48
Doctor Who, 76, 139
dokumentacja Pythona, 30, 77
drop_duplicates(), 236
dropna(), 236
drzewo katalogów, 38
dtype, 215, 230
dwukropek, *Patrz* operator zakresu
dziedziczenie, 90

E

EasyGui, 202
easygui0.py, 202
easygui1.py, 203
edytory do Pythona, 21
endswith(), 120
EOFError, 107
etc (katalog), 40
Excel, 233
    import do Pandas, 235
    import z Pandas, 249
excel0.py, 245
excel1.py, 248
excel2.py, 249
excel3.py, 249
except, 86, 104
Exception (wyjąteK), 107
exec_linux.py, 111
exec_linux2.py, 112
exec_win.py, 110
exp(), 60
extend(), 125

F

f" (notacja), 63
fabs(), 60
fc (polecenie), 44
fileopenbox(), 207
filesavebox(), 208
fillna(), 236
finally, 105
find(), 120
`findall()`, 257
findstr, 252
float(), 67
float16, 215
float32, 215
float64, 215
FloatingPointError, 107
floor(), 60
folderdel.py, 171
for (pętla), 68
formatowanie
    kodu, 48
    liczb, 65

formatowanie.py, 63
formularze, 206
funkcje, 70
  matematyczne, 60
funkcje.py, 71, 72
funkcje2.py, 73

G

get(), 141
GitHub, 32
gmtime(), 183
grep, 253
groupby(), 240
GUI, 201
Guido Van Rossum, 11

H

Harward (Uniwersytet), 259
help, *Patrz* argparse
hermetyzacja, 79
hex(), 50
histogram, 197, 224
histogram(), 224
home (katalog), 40
hstack(), 219

I

ICT, 11
IDLE, 22
import, *Patrz* moduły
import danych, 233
ImportError, 107
in, 118, 124, 141
IndentationError, 107
index(), 121, 125, 131
IndexError, 107, 118
inplace, 236
input(), *Patrz* interakcja z użytkownikiem
insert(), 125
insert_cols(), 249
insert_rows(), 249
int(), 67
int16, 215
int32, 215
int64, 215
interakcja z użytkownikiem, 103
interakcje.py, 103
interakcje2.py, 108
intersection(), 135
isalnum(), 122
isalpha(), 122
isascii(), 122
isdigit(), 122
isleap(), 184, 185
islower(), 122
isspace(), 122
issubset(), 136
issuperset(), 136
istitle(), 122
isupper(), 122
IT, 11

J

Jenkins, 10

K

kalendarz.py, 184
KeyboardInterrupt, 107
keys(), 141
klasa, 75, 79
  bazowa, 90
  projektowanie, 83
klasa0.py, 80
kodowanie, 148
komentarze, 48
konstruktor, 81
konwersje napisów na liczby, 66
Kubuś Puchatek, 259

L

len(), 118, 124, 141
less, 43
lib (katalog), 40
LiczbyZespolone.py, 101
LIFO, 126
linestyle, 192
linewidth, 192
list comprehension, 130, 246
list(), 122
lista wyboru, 205
listy.py, 127, 128
ljust(), 119
load_workbook(), 245
localtime(), 183
log(), 60
logi, 251
lower(), 119
ls, 41, 96
lstrip(), 119

M

Machine Learning, *Patrz* uczenie maszynowe
macOS, 17
marker, 192
matematyczne.py, 114
Matplotlib, 189, 221
  integracja z Pandas, 242
max(), 216, 240
mean(), 216, 240
media (katalog), 40
median(), 240
MemoryError, 107
menu sterujące skryptem, 108
menu.py, 109
metoda (klasy), 79
metody, 75
Microsoft Visual C++ Redistributable, 190
miesiace.py, 130
min(), 216, 240
mkdir, 42
mnt (katalog), 40
mode(), 237, 240
moduły, 100, 113
  publikacja w Internecie, 115
more, 43
msgbox, 204
msgbox(), 204
multenterbox(), 206
multiply(), 217
multpasswordbox(), 207

N

n, *Patrz* znak nowej linii
NameError, 107
NaN, 234
napisy, 61
napisy.py, 61, 76
ndarray, 212
negative(), 217
newline, 177
normalizowanie osi wykresu, 194
notacja z kropką, 75
now(), 186
nowy.xlsx, 249
NumPy, 190, 198, 211
numpy0.py, 214
numpy1.py, 217, 219
numpy2.py, 220
numpy3.py, 220
numpyplot.py, 221
numpyplot2.py, 222, 224

O

obiekt, 75, 79
object, 230
Object Oriented Programming, *Patrz*
  podejście zorientowane obiektowo
odczyt_zapis.py, 154
okno
  komunikatu, 204
  kontynuacji, 204
OOP, *Patrz* Object Oriented Programming
open(), 177
Openpyxl, 243
operator
  dostępu, 124
  zakresu, 66
operatory
  arytmetyczne, 51
    modelowanie w klasie, 88
  bitowe, 51
  logiczne, 54
operatory.py, 53
opt (katalog), 40
os (moduł), 160
os.system, 110
OSError, 107
OverflowError, 107

P

pakiety, 113
Pandas, 227
  import danych, 233
pandas0.py, 229
pandas1.py, 231
pandas2.py, 233, 235
pandas3.py, 237
parametry, 71
  skryptów, 96
parametry1.py, 97
parametry2.py, 98
parametry3.py, 99
parser, 99
Path (klasa), 162
petle.py, 68, 69, 70
pętle, 67
pi (stała), 60
pickle, *Patrz* serializacja

pip, 20, 75
  instalacja pakietów w PyCharm, 29
pipe, 44
pliki
  binarne, 148
  dziennika, *Patrz* logi
  tekstowe, 148
  tryby otwarcia, 150
pliki.py, 163
plikidel.py, 170
plikidir.py, 165
plikimet.py, 167
plot(), 194, 211, 242
podstawa systemu, 49
pola, 79
pomiar.py, 84
pomiarv.py, 44, 91
pomocnicze.py, 114
pop(), 125, 135, 141
Popen, 111
pow(), 60
power(), 217
procedury, 70
Program Files (katalog), 40
przekierowanie do pliku, 45
przypadek elementarny, 72
pwd, 40
PyCharm, 22, 26
  instalacja pakietów, 30
PyPi, 115
Python
  instalacja, 15

R

radians(), 60
raise, 86
raport.xlsx, 244
raport2.xlsx, 247
ravel(), 219
re (moduł), 251, 256
read(), 149
reader(), 176
readline(), 150
readlines(), 150
referencje, 57
referencje.py, 58
regex, *Patrz* wyrażenia regularne
regex0.py, 256
regex1.py, 257
remove(), 125, 135
rename.py, 172
replace(), 121, 188
reset_dimensions(), 247
reshape(), 217, 218
resize(), 218
reverse(), 125
rfind(), 120
RGB, 192
rindex(), 121
rjust(), 119
rm, 42
rmdir, 42
roboczy1.py, 102
roboczy2.py, 102
roboczy3.py, 103
rok przestępny, 184
root (katalog), 40
rsplit(), 121
rstrip(), 120
Rsyslog, 251
RuntimeError, 107

S

sbin (katalog), 40
scenariusz.py, 102
Scikit-Learn, 13
search(), 257
seek(), 152
Seksmisja, 168
selektor wyboru pliku, 207
self, 81
serializacja, 157
Series, 228
set comprehension, 138
setter, 85
shutil (moduł), 172
shutilPath.py, 173
silnia, 72
silnia (definicja), 72
sin(x), 60
skoroszyt, 244
sleep(n), 184
slice, *Patrz* wycinek
slowniki.py, 142
słowniki, 139
słowniki_wstep.py, 139
smoke test, 10
sort(), 125, 216
sort_values(), 240
split (metoda), 67
split(), 121, 156
sqrt(), 60, 217
startswith(), 120
stos, 125

str(), 67
strip(), 120
strptime(), 186
Sublime Text, 21
subprocess (moduł), 110
subtract(), 217
sudo, 252
sum(), 216, 240
super(), 92
swapcase(), 120
symetric_difference(), 136
SyntaxError, 107
sys (moduł), 97
sys.argv, 97
syslog, 251
system
  dwójkowy, 49
  dziesiętny, 49
  liczbowy, 49
  pozycyjny, 49
SystemError, 107
Sztuczna Inteligencja, 11

Ś

ścieżka, 161
ścieżka dostępowa, 38
  bezwzględna, 39
  względna, 39

T

t, *Patrz* tabulator
TabError, 107
tabulator, 119
tan(x), 60
TensoFlow, 13
Terminal, 36
test2.xlsx, 249
time (moduł), 182
time(), 182, 183
timedelta, 230
timezone, 184
title(), 120
Tkinter, 209
tmp (katalog), 40
touch(), 169
touchme.py, 169
transpose(), 219
trunc(), 60
try, 86, 104
tuple, 131
  modyfikacja, 132
tuple.py, 131, 132
type, 44, 252
TypeError, 107
typy danych, 56
tzname, 184

U

uczenie maszynowe, 11
uint16, 215
uint32, 215
uint64, 215
uint8, 215, 238
ukośnik, 147
ukośnik pojedynczy, 119
unfunc, 216
Unicode, 148
union(), 135
UNIX (czas narodzin), 182
update(), 141
upper(), 119
uprawnienia, 42
uruchamianie programów, 42
Users (katalog), 40
usr (katalog), 40
utcnow(), 186
UTF-8!, 148
uzyjmodulu.py, 114

V

ValueError, 107, 121, 176, 238
values(), 141
var (katalog), 40
Vim, 21
vstack(), 219

W

warunki.py, 55
wcięcia, 48
while (pętla), 68
wielolinijkowe napisy, 48
wieloznacznik, 166
wiersz linii poleceń, 36
wildcard, *Patrz* wzorzec poszukiwań, *Patrz* wieloznacznik
Windows (katalog), 40
witamy.py, 19, 45
właściwości, 84

writer(), 177
writerow(), 177
wycinek, 118
wycinki, 220
wyjatekEOF.py, 106
wyjatki2.py, 104
wyjątki, 104
wyjątki.py, 106
wykres
  formatowanie, 192
wykres słupkowy, 196
wykres0.py, 191
wykres1.py, 193
wykres2.py, 194
wykres3.py, 196
wykres4.py, 197
wykrescsv.py, 198
wykresy, 191
wykresy wielokrotne, 194, 222
wyrażenia
  regularne, 255
  warunkowe, 54
wywołanie programu w skrypcie, 110
wzorzec
  poszukiwań, 41
  wyszukiwania, 255

Y

ynbox, 204
ynbox(), 204

Z

zakresy.py, 66
zapis szesnastkowy, 50
zapisCSHV.py, 179
zapisCSV.py, 177
zasiegi.py, 74
zasięg zmiennych, 74
zbiory, 134
zbiory.py, 134
ZeroDivisionError, 107
zespolone.py, 101
zmienne, 56
  zasięg, 74
zmienne.py, 56
znak nowej linii, 61, 119
znaki specjalne, 119

Też masz wrażenie, że Python jest ostatnimi czasy dosłownie wszędzie? Nic dziwnego – to najbardziej uniwersalny i przystępny język programowania, jaki kiedykolwiek powstał!

Jeśli chcesz poznać go od podstaw, sięgnij po odpowiedni podręcznik – taki jak ta książka! To wydanie przeznaczone dla użytkowników Linuxa (także macOS) i Windowsa; ewentualne cechy specyficzne dla konkretnych systemów są na bieżąco wyjaśniane w tekście. Zawiera zagadnienia ukierunkowane na praktyczne potrzeby testerów oprogramowania, którzy pragną wkroczyć w magiczny świat automatyzacji zadań. Została napisana przez autora wielu książek z dziedziny programowania w okresie, gdy kierował zespołem testerów w dziale rozwoju oprogramowania dużej firmy telekomunikacyjnej, realizującym zaawansowane testy manualne i automatyczne.

Opis języka opiera się na najnowszej specyfikacji języka (wersja 3.9x lub wyższe).

- Środowisko Pythona i polecane pakiety IDE
- Z terminalem Windows i Linux za pan brat
- Systemy liczbowe i kodowanie dla nieinformatyków
- Błyskawiczny kurs języka
- Typy i struktury danych bez tajemnic
- Interakcja z użytkownikiem
- Zapis i odczytywanie danych z plików
- Tajniki plików CSV
- Programowanie obiektowe w Pythonie
- Własne biblioteki (moduły)
- Analiza danych z NumPy i Pandas
- Tworzenie wykresów w Matplotlib
- Python i Excel
- Analiza zawartości logów i wyrażenia regularne (regex)
- Proste aplikacje okienkowe z EasyGUI

Programuj, uruchamiaj, automatyzuj – przekonaj się, jak dużo oferuje Python!

Piotr Wróblewski – doświadczony kierownik projektów informatycznych, wcześniej także Product Manager oraz kierownik zespołu testerów oprogramowania w dużej firmie telekomunikacyjnej. Od 1992 roku stały współpracownik wydawnictwa Helion. Autor wielu cenionych książek.

Helion

helion.pl

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

ISBN 978-83-283-8404-0

Cena: 69,00 zł