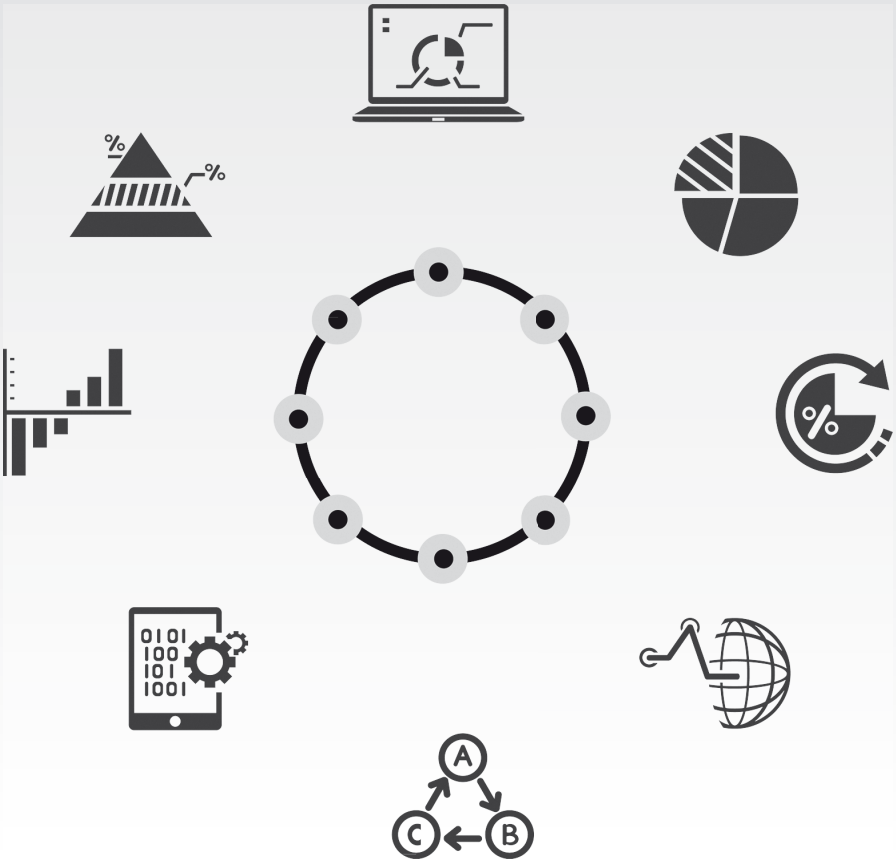
Marek Gągolewski
Maciej Bartoszuk
Anna Cena

Przetwarzanie i analiza danych w języku Python



PWN

Przetwarzanie i analiza danych w języku Python

Marek Gągolewski
Maciej Bartoszuk
Anna Cena

Przetwarzanie i analiza danych w języku Python

PWN

Projekt okładki **Hubert Zacharski**

Ilustracja na okładce **shutterstock/phipatbig**

Wydawca **Łukasz Łopuszański**

Redaktor prowadzący **Iwona Lewandowska**

Redaktor **Ewa Ławrynowicz**

Koordynator produkcji **Anna Bączkowska**

Skład i łamanie **FixPoint**

ISBN: 978-83-01-18940-2

Wydanie I
Warszawa 2016

Wydawnictwo Naukowe PWN SA
02-460 Warszawa, ul. Gottlieba Daimlera 2
tel. 22 69 54 321, faks 22 69 54 288
infolinia 801 33 33 88
e-mail: pwn@pwn.com.pl; reklama@pwn.pl
www.pwn.pl

Druk i oprawa: OSDW Azymut Sp. z o.o.

SPIS TREŚCI

Przedmowa XI

I Podstawy języka Python

1. Wprowadzenie 3

1.1. Język i środowisko Python 3
1.1.1. Instalacja dystrybucji środowiska Python 3
1.1.2. Instalacja pakietów 5
1.2. Notatniki Jupyter 7
1.2.1. Tryby pracy 7
1.2.2. Najważniejsze skróty klawiszowe 10
1.2.3. Podstawy języka Markdown 10
1.3. Pierwsze kroki w języku Python 12

2. Typy skalarne 16

2.1. Liczby 16
2.1.1. Operatory arytmetyczne 18
2.1.2. Konwersja typów 21
2.1.3. Tworzenie obiektów nazwanych 22
2.1.4. Funkcje wbudowane 23
2.1.5. Pola i metody 24
2.1.6. Arytmetyka zmiennopozycyjna 25
2.2. Wartości logiczne 26
2.2.1. Operatory relacyjne 27
2.2.2. Operatory logiczne 28
2.3. Napisy 28
2.3.1. Tworzenie napisów 28
2.3.2. Podstawowe operacje na napisach 30

3. Typy sekwencyjne i iterowalne 32

3.1. Podstawowe rodziny obiektów typu sekwencyjnego 33
3.1.1. Listy i krotki 33
3.1.2. Zakresy 35
3.1.3. Napisy 35

3.2. Zarządzanie elementami . . . 35
3.2.1. Wybieranie elementów . . . 35
3.2.2. Modyfikacja elementów . . . 38
3.2.3. Dodawanie i usuwanie elementów . . . 39
3.2.4. Kopiowanie referencji, kopiowanie płytkie a głębokie . . . 41
3.3. Obiekty iterowalne . . . 45
3.4. Działania na obiektach iterowalnych i typu sekwencyjnego . . . 47
3.4.1. Podstawowe metody i funkcje . . . 47
3.4.2. Krotki identyfikatorów po lewej stronie operatora przypisania . . . 50
3.4.3. Wyrażenia listotwórcze i generatory . . . 51
3.4.4. Formatowanie napisów . . . 54

4. Słowniki i zbiory . . . 56

4.1. Słowniki . . . 56
4.1.1. Tworzenie słowników . . . 56
4.1.2. Podstawowe metody i funkcje . . . 58
4.2. Zbiory . . . 61
4.2.1. Tworzenie zbiorów . . . 61
4.2.2. Podstawowe metody i funkcje . . . 62

5. Instrukcje sterujące . . . 64

5.1. Instrukcja warunkowa . . . 64
5.2. Pętle . . . 66
5.2.1. Pętla `while` . . . 66
5.2.2. Pętla `for` . . . 67
5.2.3. Instrukcje `break` i `continue` oraz blok `else` w pętlach . . . 69
5.3. Obsługa wyjątków . . . 73
5.3.1. Zgłaszanie wyjątków . . . 74
5.3.2. Rodzaje wyjątków . . . 74
5.3.3. Wychwytywanie wyjątków . . . 75

6. Funkcje . . . 77

6.1. Definiowanie funkcji . . . 77
6.1.1. Dokumentowanie funkcji . . . 78
6.1.2. Wartość zwracana . . . 79
6.1.3. Wyrażenia lambda . . . 80
6.2. Parametry i argumenty . . . 81
6.2.1. Sposób przekazywania argumentów . . . 81
6.2.2. Sprawdzanie poprawności argumentów . . . 82
6.2.3. Dopasowywanie argumentów . . . 84
6.2.4. Parametry z argumentami domyślnymi . . . 84
6.2.5. Rozpakowywanie argumentów . . . 85
6.2.6. Parametry specjalne `*args` i `**kwargs` . . . 86
6.3. Zasięg zmiennych . . . 88
6.3.1. Zmienne lokalne . . . 88
6.3.2. Zmienne globalne . . . 88
6.3.3. Zmienne nielokalne, fabryki funkcji i domknięcia . . . 90
6.4. Pakiety . . . 92

II Przetwarzanie danych

7. Wektory, macierze i inne tablice . . . 97

7.1. Tworzenie i reprezentacja tablic . . . 97
7.1.1. Funkcja `array()` . . . 98
7.1.2. Reprezentacja tablic . . . 100
7.1.3. Typ przechowywanych elementów . . . 101
7.1.4. Tworzenie tablic specjalnego rodzaju . . . 103
7.1.5. Łączenie tablic . . . 106
7.2. Podstawowe metody i funkcje . . . 108
7.2.1. Operatory arytmetyczne. Uzgadnianie kształtów . . . 108
7.2.2. Operacje relacyjne i logiczne . . . 113
7.2.3. Zwektoryzowane funkcje matematyczne . . . 115
7.2.4. Agregacja danych . . . 118
7.2.5. Inne operacje . . . 121
7.3. Indeksowanie tablic . . . 123
7.3.1. Indeksowanie wektorów . . . 123
7.3.2. Indeksowanie macierzy . . . 128
7.3.3. Indeksowanie tablic *N*-wymiarowych . . . 132
7.3.4. Wyszukiwanie indeksów elementów spełniających zadane kryteria . . . 134

8. Ramki danych . . . 137

8.1. Tworzenie ramek danych . . . 138
8.1.1. Konstruktor klasy `DataFrame` . . . 138
8.1.2. Importowanie ramek danych z plików i innych źrodeł . . . 139
8.1.3. Odczytywanie podstawowych informacji o ramkach danych . . . 140
8.2. Zmienne, czyli obiekty typu `Series` . . . 143
8.2.1. Wydobywanie poszczególnych zmiennych . . . 143
8.2.2. Tworzenie i reprezentacja zmiennych . . . 144
8.2.3. Zmienne typu data i czas . . . 145
8.2.4. Zmienne jakościowe i porządkowe . . . 146
8.3. Etykiety, czyli obiekty typu `Index` . . . 150
8.3.1. Etykietowanie wierszy i kolumn . . . 151
8.3.2. Etykiety hierarchiczne . . . 152
8.4. Indeksowanie zmiennych i ramek danych . . . 154
8.4.1. Wybór elementów pojedynczej zmiennej . . . 154
8.4.2. Wybór podzbioru wierszy i kolumn ramki danych . . . 160
8.5. Wybrane operacje . . . 164
8.5.1. Dodawanie oraz usuwanie kolumn i wierszy . . . 164
8.5.2. Przekształcanie zmiennych . . . 166
8.5.3. Podsumowania ramek danych i zmiennych . . . 168
8.5.4. Sortowanie ramek danych . . . 172
8.5.5. Zmiana kształtu ramek danych . . . 173
8.5.6. Obserwacje brakujące . . . 176

9. Przetwarzanie napisów . . . 179

9.1. Operacje na pojedynczych napisach . . . 179
9.1.1. Podstawowe stałe napisowe i operacje na pojedynczych znakach . . . 180

9.1.2. Wyszukiwanie ustalonego wzorca . . . 182
9.1.3. Translacja znaków . . . 183
9.1.4. Sprawdzanie, czy wszystkie znaki należą do podanej kategorii . . . 184
9.1.5. Dzielenie i sklejanie tekstu . . . 184
9.2. Wyszukiwanie wzorca przy użyciu wyrażeń regularnych . . . 185
9.2.1. Definiowanie wyrażeń regularnych . . . 186
9.2.2. Przegląd funkcji . . . 188
9.2.3. Wydzielone podwyrażenia i odwołania do nich . . . 189
9.3. Zwektoryzowane operacje na obiektach `Index` i `Series` . . . 190

10. Przetwarzanie plików i zasobów w internecie . . . 196

10.1. Operacje na drzewie katalogów . . . 196
10.1.1. Ścieżki dostępu . . . 196
10.1.2. Wyszukiwanie plików na dysku . . . 198
10.2. Przetwarzanie plików . . . 200
10.2.1. Otwieranie pliku w różnych trybach . . . 200
10.2.2. Odczytywanie zawartości pliku . . . 202
10.2.3. Zapisywanie danych do pliku . . . 203
10.2.4. Serializacja obiektów . . . 204
10.2.5. Popularne formaty plików . . . 205
10.3. Pozyskiwanie danych ze stron internetowych . . . 208
10.3.1. Wydobywanie tabel w postaci ramek danych . . . 209
10.3.2. Ręczne przetwarzanie kodu źródłowego strony . . . 209
10.3.3. Parsowanie kodu HTML i wydobywanie pojedynczych elementów . . . 211

11. Dostęp do baz danych . . . 215

11.1. Przykładowa baza danych: `nycflights13` . . . 215
11.2. Obsługa baz danych . . . 218
11.2.1. Połączenie z bazą danych . . . 218
11.2.2. Eksportowanie danych do bazy . . . 218
11.2.3. Odczytywanie danych z bazy . . . 219
11.2.4. Funkcje z pakietu `pandas` . . . 220
11.3. Ćwiczenia . . . 221
11.3.1. Wybór unikatowych podzbiorów kolumn . . . 222
11.3.2. Agregacja danych w podgrupach . . . 223
11.3.3. Filtrowanie danych wejściowych i wyników . . . 226
11.3.4. Sortowanie wyników . . . 230
11.3.5. Operacje teoriomnogościowe . . . 232
11.3.6. Złączenia . . . 234

III Analiza danych

12. Wizualizacja danych . . . 239

12.1. Rysowanie podstawowych obiektów . . . 240
12.1.1. Łamane . . . 240
12.1.2. Punkty i różne symbole . . . 241
12.1.3. Wielokąty . . . 242
12.1.4. Adnotacje tekstowe . . . 243

12.2. Parametry graficzne . . . 244
12.2.1. Sposoby kreślenia punktów i odcinków . . . 244
12.2.2. Sposoby określania barw . . . 244
12.2.3. Napisy formatujące . . . 246
12.2.4. Ustawienia osi . . . 247
12.3. Rysunki jako kombinacje obiektów podstawowych . . . 248
12.3.1. Wiele obiektów na jednym wykresie . . . 248
12.3.2. Legenda . . . 250
12.3.3. Wiele wykresów na jednej stronie . . . 251
12.4. Graficzna prezentacja danych . . . 255
12.4.1. Wybrane wykresy dla danych jakościowych . . . 255
12.4.2. Wybrane wykresy dla danych ilościowych . . . 258
12.4.3. Wybrane wykresy dla funkcji dwuwymiarowych . . . 262

13. Wnioskowanie statystyczne . . . 265

13.1. Wybrane rozkłady prawdopodobieństwa . . . 265
13.1.1. Podstawowe rodziny rozkładów . . . 265
13.1.2. Generowanie liczb pseudolosowych . . . 273
13.2. Estymacja parametrów i charakterystyk rozkładów . . . 275
13.2.1. Estymacja punktowa . . . 276
13.2.2. Estymacja przedziałowa . . . 278
13.3. Wykorzystanie testów statystycznych w analizie danych . . . 280
13.3.1. Testy zgodności . . . 281
13.3.2. Testy parametryczne . . . 290
13.3.3. Testy nieparametryczne . . . 295

14. Wybrane algorytmy uczenia maszynowego . . . 298

14.1. Przykładowy zbiór danych: `winequality` . . . 298
14.2. Analiza regresji . . . 300
14.2.1. Regresja liniowa . . . 301
14.2.2. Ocena jakości dopasowania modelu . . . 304
14.2.3. Model wielomianowy . . . 306
14.2.4. Wybór zmiennych do modelu . . . 307
14.3. Klasyfikacja . . . 310
14.3.1. Metoda k-najbliższych sąsiadów . . . 312
14.3.2. Ocena jakości klasyfikatora . . . 312
14.3.3. Drzewa decyzyjne i lasy losowe . . . 315
14.3.4. Porównanie krzyżowe . . . 318
14.4. Analiza skupień . . . 320
14.4.1. Algorytm k-średnich . . . 320
14.4.2. Hierarchiczna analiza skupień . . . 326

IV Tworzenie własnego oprogramowania

15. Moduły, pakiety i skrypty . . . 331

15.1. Projekty wielomodułowe . . . 331
15.1.1. Środowisko programistyczne Spyder . . . 331
15.1.2. Tworzenie i ładowanie modułów . . . 332

15.1.3. Tworzenie i ładowanie pakietów 335
15.1.4. Ścieżki wyszukiwania modułów i pakietów 336
15.2. Skrypty 336
15.2.1. Uruchomienie skryptu z poziomu powłoki 337
15.2.2. Przekazywanie argumentów 338
15.2.3. Skrypty a moduły. Testy jednostkowe 339

16. Programowanie obiektowe 343

16.1. Klasy i relacje między nimi 344
16.1.1. Definiowanie klasy 344
16.1.2. Dziedziczenie 346
16.2. Metody 348
16.2.1. Przeciążanie metod. Polimorfizm 348
16.2.2. Metody i pola statyczne 350
16.2.3. Metody specjalne 351
16.3. Pola 357
16.3.1. Definiowanie z góry ustalonych pól w klasie 357
16.3.2. Pola prywatne, chronione i publiczne 358

Bibliografia 361

Skorowidz 363

PRZEDMOWA

Żyjemy w fascynującej dobie przeciążenia informacyjnego: dla korporacji, instytucji i zainteresowanych jednostek zarówno samo zdobycie różnorakich danych, jak i późniejsze ich przechowywanie nie stanowi już właściwie żadnego problemu – jeśli tylko dysponują odpowiednimi środkami. Wyzwaniem pozostaje jednak przetworzenie tego oceanu nic niemówiących ciągów bitów na użyteczną wiedzę, np. plan marketingowy firmy, zoptymalizowany pod różnymi względami sposób rekomendowania treści użytkownikom portalu internetowego, taktykę inwestowania na giełdzie, dalszy kierunek badań R&D. Z tego rodzaju wyzwaniami mierzy się m.in. względnie nowa, ale prężnie rozwijająca się dziedzina *inżynierii i analizy danych* (ang. *data science*), która wymaga od zajmującej się nią profesjonalistów nie tylko dziedzinowej wiedzy eksperckiej oraz szeroko pojętej kreatywności i ciekawości świata, ale także kompetencji matematycznych (w szczególności statystycznych) oraz umiejętności programistycznych; por. rys. 0.1.

Na rynku dostępnych jest wiele narzędzi do obliczeń analizodanowych, jednak w ostatnim czasie obserwujemy, że wielu specjalistów coraz chętniej sięga po otwarte

Rysunek 0.1. Trzy główne obszary kompetencji specjalistów inżynierii i analizy danych

i wolnodostępne rozwiązania. Aby oprogramowanie spełniało swoją funkcję w wyróżnionym przez nas obszarze zastosowań, musi zawierać pokaźny zestaw najbardziej pożytecznych, sprawdzonych, gotowych do użycia metod, służących m.in. do sporządzania wykresów, wyznaczania modeli regresji i klasyfikacji czy weryfikowania hipotez statystycznych. Co ważne, musi być ono oparte także na „pełnowymiarowym" *języku programowania*, tak by korzystający z niego nie czuł się w żaden sposób ograniczony – prawdziwe wyzwania *data science* bardzo rzadko wpisują się w proste szablony. Dzięki temu możliwe jest także automatyzowanie procesów przetwarzania danych i sprawianie, by przeprowadzane analizy były odtwarzalne.

O jednym z takich niezmiernie popularnych narzędzi – środowisku R (zob. np. [16]) – mówi się, że zostało stworzone przez statystyków dla statystyków. Mimo że R oferuje coraz lepszą łączność ze wszystkimi popularnymi systemami baz danych, wsparcie dla systemów *big data* (HDFS, Spark itd.), możliwość tworzenia rozszerzeń w językach Java i C++, bardzo rzadko jest on doceniany przez osoby niemające wykształcenia okołoanalizodanowego. W szczególności — z racji tego, że semantyka języka R została zainspirowana różnymi pochodnymi języka Lisp, który współcześnie nie jest już zbyt często używany — obserwujemy, że trudno jest do niego przekonać informatyków: ich opinia jest o tyle ważna, że np. implementowane przez specjalistów *data science* modele predykcyjne stają się potem częścią większych rozwiązań projektowych.

Środowisko Python jest z kolei oparte na języku o znacznie szerszych zastosowaniach. Stworzone przez programistów dla programistów pozwala na tworzenie wszelkiej maści projektów informatycznych – od szybkich prototypów rozwiązań (zgodnie z ideą *rapid development*) po duże, wielomodułowe aplikacje składające się z kilkuset klas. Jego wysoka przydatność jest powszechnie znana od wielu lat w takich obszarach, jak programowanie aplikacji sieciowych, internetowych i mobilnych, przetwarzanie obrazów i sygnałów audio, tworzenie gier komputerowych itp. Co ważne, od jakiegoś czasu w szerokiej gamie powiązanych ze sobą pakietów i bibliotek możemy odnaleźć wiele nowoczesnych i wydajnie zaimplementowanych algorytmów uczenia maszynowego (ang. *machine learning*), statystycznych systemów uczących się, wizualizacji danych itd. Wszystko to sprawia, że to właśnie Python staje coraz częściej narzędziem z wyboru dla specjalistów *data science* – ma ono bowiem ogromne możliwości.

Wśród dostępnych na polskim rynku wydawniczym pozycji poświęconych wprowadzeniu do programowania w „bazowym" języku Python możemy wymienić m.in. [1, 8, 18, 27, 34, 35, 46]. Zainteresowani szeroko pojętą analizą danych do tej pory zmuszeni byli już jednak odwoływać się do lepszych lub gorszych angielskojęzycznych tytułów [4, 9, 20, 26, 40, 41, 43, 48]. Niniejsza książka jest pierwszą polską pozycją poświęconą w całości przetwarzaniu i analizie danych z wykorzystaniem środowiska Python.

Cel książki i jej adresaci. Książka stawia sobie za cel przygotować Czytelnika do samodzielnego przeprowadzenia procesu analizy danych, od pobrania i załadowania zbioru danych, poprzez jego wstępne przetworzenie i wyczyszczenie, aż po samą analizę.

Czytelnik będzie potrafił dokonać wizualizacji danych oraz przedstawić wyniki analizy w formie raportów. Ponieważ wierzymy, że lepiej dać przysłowiową wędkę niż złowioną rybę, kładziemy nacisk na dokładne omówienie samego języka Python 3 i najważniejszych pakietów towarzyszących, tak by Czytelnik był przygotowany na twórcze mierzenie się z nowymi problemami oraz dalsze samodzielne zgłębianie literatury przedmiotu. Wiemy, że pewne rozwiązania, które stworzy Czytelnik, będą przeznaczone do wielokrotnego użytku i tym samym będą zasługiwać na wdrożenie w ramach większych projektów informatycznych. Z tego powodu omawiamy także zestaw dobrych praktyk z zakresu inżynierii oprogramowania.

Książka jest przeznaczona dla szerokiego grona odbiorców, m.in. (obecnych i przyszłych) analityków danych, *data scientists*, specjalistów *business intelligence*, naukowców, badaczy i programistów. Szczególnie polecamy ją:

— Osobom, które już mają wiedzę teoretyczną i umiejętności w zakresie wizualizacji danych, statystyki lub uczenia maszynowego, które wykorzystywały oprogramowanie typu R, SAS czy SPSS w analizie danych i które pragną wreszcie poznać, jak wykonać podobne czynności w środowisku Python.

— Programistom języka Python zainteresowanym inżynierią i analizą danych w całej swej okazałości. Proponowany kurs zapoznaje Czytelnika z najistotniejszymi technikami, dając przy tym inspirację i podstawy do dalszego – samodzielnego już – zgłębiania wiedzy teoretycznej i rozwoju umiejętności praktycznych. Pozwala też zrozumieć wyzwania stające przed analitykami danych i znaleźć z nimi „wspólny język".

— Osobom, które znają podstawy programowania w jakimś innym języku, a które chcą poznać język Python w kontekście, którego nie oferują inne podręczniki. Książka bowiem zawiera całościowy kurs języka – problemy analizy danych dla niektórych mogą być po prostu niewinnym pretekstem do wykonania całej serii przyjemnych ćwiczeń. Ciekawe zbiory danych prowokują do stawiania pytań i poszukiwania niebanalnych odpowiedzi, do których droga prowadzi przez programowanie. Po zapoznaniu się z przedstawionym tutaj materiałem Czytelnika nie zdziwi żadna zawiłość ani cecha języka, będzie on w stanie swobodnie posługiwać się dokumentacją różnych pakietów, a także samodzielnie utworzyć większy fragment oprogramowania.

Pisząc ten podręcznik, mieliśmy również na uwadze potrzeby studentów i wykładowców kierunków takich jak matematyka, statystyka, informatyka, fizyka czy ekonometria – jako podręcznik wspomagający zajęcia ze statystyki lub uczenia maszynowego bądź będący podstawą kursu wprowadzającego do szeroko rozumianej inżynierii i analizy danych. Czerpaliśmy z doświadczenia zdobytego w trakcie prowadzenia zajęć na Wydziale Matematyki i Nauk Informacyjnych Politechniki Warszawskiej oraz szkoleń z cyklu *Data Science Retreat* w Berlinie.

Struktura książki. Materiał podzieliliśmy na cztery następujące części. Ufamy, że ten przejrzysty układ pozwoli nie tylko lepiej uporządkować przyswajaną wiedzę, ale i później – każdorazowo w razie potrzeby – łatwo wyszukiwać potrzebne informacje.

Podstawy języka Python 3. Zaczynamy od omówienia instalacji dystrybucji Anaconda i sposobów pracy z notatnikami Jupyter w rozdz. 1. Następnie omawiamy w wyczerpujący sposób podstawy „bazowego" języka Python – w każdym razie w zakresie potrzebnym do rozpoczęcia prawdziwej przygody z *data science*. W szczególności interesują nas najważniejsze typy danych: wartości skalarne (rozdz. 2), listy, krotki i inne typy sekwencyjne oraz iterowalne (rozdz. 3), słowniki i zbiory (rozdz. 4), a także instrukcje sterujące (rozdz. 5) i sposoby definiowania własnych funkcji (rozdz. 6).

Przetwarzanie danych. W drugiej części zajmujemy się zagadnieniami związanymi ze wstępnym przetwarzaniem danych i przygotowaniem ich do analizy. Omawiamy szczegółowo pakiet numpy, który udostępnia wektory, macierze i inne *n*-wymiarowe tablice a także szeroką gamę metod i funkcji operujących na nich (rozdz. 7). Dalej skupiamy się na opartym na numpy pakiecie pandas, przy którego użyciu możemy reprezentować i przekształcać rekordy zapisane w postaci tabelarycznej (rozdz. 8). Nie zapominamy przy tym o innych ważnych zagadnieniach: przetwarzaniu napisów i wydobywaniu wiedzy z informacji tekstowych (rozdz. 9), obsłudze plików i automatycznym zbieraniu informacji z internetu (rozdz. 10), a także łączeniu się z bazami danych SQL (rozdz. 11). Co więcej, na zakończenie tej partii materiału przedstawiamy zestaw kilkudziesięciu ćwiczeń, które są poświęcone najczęściej wykonywanym w praktyce operacjom na ramkach danych, m.in. wyszukiwaniu informacji, przekształceniu zmiennych, filtrowaniu wierszy i kolumn, agregacji zmiennych w podgrupach utworzonych przez kombinacje wielu czynników oraz złączaniu tabel.

Analiza danych. W trzeciej części nasza uwaga jest skupiona na szeroko pojętej analizie danych, czyli na różnorakich metodach, które pozwalają przekuwać surowe informacje na użyteczną wiedzę. Najpierw poznajemy pakiety matplotlib i seaborn, na których podstawie będziemy dokonywać wizualizacji różnych aspektów udostępnionych nam danych oraz wyników przeprowadzanych analiz (rozdz. 12). Następnie przechodzimy do opisu dostępnych w środowisku Python metod statystycznych (rozdz. 13) – w szczególności problemów estymacji nieznanych parametrów i charakterystyk rozkładów oraz weryfikacji hipotez. Dzięki nim będziemy potrafili odpowiadać na pewne istotne pytania w sytuacji, gdy mamy do czynienia z *nie-wielkimi* próbkami, np. czy wpływ określonego czynnika na zachowanie się pewnej zmiennej jest rzeczywiście istotny. Z kolei w rozdz. 14 omawiamy trzy najważniejsze grupy algorytmów maszynowego uczenia się: regresji, klasyfikacji i analizy skupień. Przy ich użyciu możemy modelować różne rodzaje zależności między zmiennymi, przewidywać wartości kluczowych charakterystyk dla jeszcze niezaobserwowanych próbek oraz dokonywać automatycznej segmentacji (podziału) zbioru danych na ciekawe podgrupy.

Tworzenie własnego oprogramowania. Ostatnią część książki poświęcamy zagadnieniom z dziedziny inżynierii oprogramowania – dobrze działający proces przetwarzania czy modelowania danych nierzadko należy wdrożyć jako część większego projektu informatycznego. I tak w rozdz. 15 poznajemy sposoby tworzenia własnych modułów, pakietów i skryptów, a w 16 – własnych klas, czyli nowych typów danych.

Dzięki nim możemy efektywniej panować nad złożonością pisanych przez nas programów i wygodniej dzielić się efektami naszej pracy z innymi.

WAŻNE

Materiały uzupełniające do książki, m.in. zbiory danych, przykładowe skrypty i erraty, udostępniliśmy na stronie github.com/gagolews/Analiza_danych_w_jezyku_Python/.

Podziękowania. Chcielibyśmy serdecznie podziękować naszym współpracownikom i przyjaciołom: Barbarze Żogale-Siudem (Instytut Badań Systemowych PAN) i Grzegorzowi Siudemowi (Wydział Fizyki Politechniki Warszawskiej) za liczne uwagi, komentarze i erraty do wstępnej wersji niniejszej książki. Dziękujemy także naszym wyjątkowym studentom na Wydziale Matematyki i Nauk Informacyjnych PW – w szczególności Natalii Potockiej, Martynie Śpiewak oraz Małgorzacie Dobkowskiej – za uczestnictwo w jej „beta testach" oraz p. Izabeli Mika (Instytut Podstawowych Problemów Techniki PAN) za wiele cennych porad redakcyjnych i językowych.

Marek Gągolewski
Maciej Bartoszuk
Anna Cena

Warszawa, sierpień 2016 r.

Część I

PODSTAWY JĘZYKA PYTHON

WPROWADZENIE 1

1.1. Język i środowisko Python

Python jest niezwykle popularnym językiem programowania wysokiego poziomu. Jego twórca – Guido van Rossum – zaczął pracować nad nim pod koniec lat osiemdziesiątych XX w. Warto zwrócić uwagę na następujące jego cechy. Jest to język:

- *interpretowany* – programy nie muszą być kompilowane przed uruchomieniem; dzięki temu możemy wykonywać instrukcje kolejno jedna po drugiej i od razu śledzić wyniki przeprowadzanych obliczeń;
- *zorientowany obiektowo* – łatwo w nim nie tylko tworzyć nowe typy danych, ale i stosunkowo duże projekty informatyczne;
- *ogólnego zastosowania* – jest chętnie używany we wszelkich możliwych obszarach praktyki.

Warto podkreślić, że język Python jest częścią całego „ekosystemu", na który składa się, oprócz samego interpretera, m.in. bardzo rozbudowana biblioteka *pakietów* rozszerzających jego możliwości. Już przysłowiowy rzut oka na repozytorium PyPI (od ang. *Python Package Index*, zob. pypi.python.org/) wystarczy, by przekonać się, że wśród prawie 100 000 wymienionych tam projektów każdy może znaleźć coś dla siebie.

W szczególności, nas tutaj najbardziej będą interesować pakiety, przy użyciu których będziemy w stanie przeprowadzać obliczenia analizodanowe. Zwróćmy uwagę, że dziedzina ta jest w ostatnich latach wyjątkowo intensywnie rozwijanym obszarem zastosowań języka Python. Aby uzyskać do nich wygodny dostęp, powinniśmy wybrać najodpowiedniejszą dla nas dystrybucję tego środowiska.

1.1.1. Instalacja dystrybucji środowiska Python

W internecie można znaleźć wiele dystrybucji środowiska Python zawierających interpreter samego języka oraz zestaw instalowanych wraz z nim pakietów. Na przykład użytkownicy systemu operacyjnego Linux mają do niego dostęp z poziomu właściwego im menedżera oprogramowania (yum, apt-get itd.).

W niniejszym opracowaniu zalecamy jednak korzystanie z *dystrybucji Anaconda*. Stosowny instalator – dla systemu operacyjnego Windows, OS X lub Linux – jest dostępny do pobrania ze strony internetowej www.continuum.io/downloads/. Warto zauważyć, że Anaconda jest całkowicie darmową dystrybucją środowiska Python,

którą można wykorzystywać także w zastosowaniach komercyjnych. Jest ona „skrojona" na potrzeby przetwarzania i analizy danych – zawiera bowiem kilkaset najbardziej przydatnych pakietów rozszerzających możliwości tego języka.

ZADANIE 1.1. Zainstaluj środowisko Python w wersji 3.x zawarte w dystrybucji Anaconda. Jako katalog docelowy wybierz np. `/opt/anaconda3/` (Linux, OS X) lub `c:\anaconda3\` (Windows).

Oczywiście instalacja dystrybucji Anaconda nie musi odbywać się z konta administratora. Można ją też skopiować np. do katalogu domowego użytkownika. Bez straty ogólności zakładamy jednak tutaj, że katalog docelowy był taki, jak podany wyżej.

Aby sprawdzić zainstalowaną wersję interpretera języka Python, wywołujemy następujące polecenie w powłoce (np. bash, zsh; Linux, OS X):

```
$ /opt/anaconda3/bin/python -V
Python 3.5.1 :: Anaconda custom (64-bit)
```

lub wierszu poleceń (*Menu Start → Anaconda3 → Anaconda Prompt* w Windows):

```
[anaconda3] > python -V
Python 3.5.1 :: Anaconda 4.0.0 (32-bit)
```

Mimo że ze środowiskiem Python będziemy pracować najczęściej w trybie interaktywnym za pośrednictwem notatników Jupyter (zob. podrozdz. 1.2), pierwsze polecenie wykonajmy przy użyciu klasycznego interpretera. I tak funkcja `print()` służy do wyświetlenia na konsoli napisu[1] podanego jej jako argument.

```
$ /opt/anaconda3/bin/python
Python 3.5.1 |Anaconda custom (64-bit)| (default, Dec  7 2015, 11:16:01)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> print("Dobry wieczór uczniowie, \
... witajcie na drugiej lekcji języka włoskiego.")
Dobry wieczór uczniowie, witajcie na drugiej lekcji języka włoskiego.
```

Podobnie będzie pod Windows:

```
[anaconda3] > python
Python 3.5.1 |Anaconda 4.0.0 (32-bit)| (default, Mar  4 2016, 15:28:01)
[MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> print("Dobry wieczór uczniowie, \
... witajcie na drugiej lekcji jezyka wloskiego.")
Dobry wieczór uczniowie, witajcie na drugiej lekcji języka włoskiego.
```

[1] Większość cytatów pochodzi ze skeczów grupy *Monty Python* w tłumaczeniu Tomasza Beksińskiego.

W tym momencie, wpisując polecenie `exit()` lub naciskając kombinację klawiszy ⟨CTRL+D⟩, zakończymy pracę z interpreterem.

CIEKAWOSTKA

Pobierając instalator dystrybucji Anaconda, z pewnością zauważyliśmy, że mogliśmy wybrać także wersję 2.7 interesującego nas środowiska. Opublikowana w 2008 r. wersja 3.0 języka nie jest całkowicie kompatybilna z rodziną 2.x – jej autorzy wprowadzili wówczas szereg bardzo głębokich zmian. Mimo że Python 2.7 jest jeszcze aktualizowany (wprowadzane są jednak tylko poprawki błędów) i używany przez niektórych programistów, to właśnie wersja 3.x jest bardziej godna uwagi jako ta jedynie „przyszłościowa".

Wersję 2.x najłatwiej odróżnić od 3.x, wykonując następującą instrukcję:

```
>>> print 1/2
```

Podane polecenie jest po prostu nieprawidłowe w języku Python 3.x. We wcześniejszej wersji otrzymujemy w wyniku wartość równą zeru (dzielenie całkowite). Natomiast w wersji 3.x, wykonując instrukcję analogiczną, uzyskamy:

```
>>> print(1/2)
0.5
```

O tego typu mniej lub bardziej znaczących różnicach warto pamiętać, studiując m.in. przykładowe kody dostępne w internecie.

1.1.2. Instalacja pakietów

Pakiet to zbiór funkcji, nowych typów danych i innych obiektów, które rozszerzają możliwości środowiska Python; por. rozdz. 15.

ZADANIE 1.2. Zaktualizuj wszystkie zainstalowane pakiety, wywołując w powłoce:

```
$ sudo /opt/anaconda3/bin/conda update --all
```

lub w wierszu poleceń:

```
[anaconda3] > conda update --all
```

Dystrybucja Anaconda domyślnie instaluje wiele używanych przez nas pakietów, m.in. numpy, pandas, matplotlib, scikit-learn (sklearn), statsmodels, sqlite3. Nie ma jednak wśród nich m.in. pakietu seaborn, który zawiera zestaw ciekawych metod do generowania wykresów oraz przykładowych zbiorów danych. Aby móc z niego skorzystać

np. w rozdz. 8 i 12, musimy go doinstalować ręcznie. Zróbmy to dla ćwiczenia właśnie w tym miejscu.

Dodatkowe pakiety z repozytorium PyPI instalujemy w naszym przypadku najczęściej przy użyciu programu conda.

ZADANIE 1.3. Zainstaluj pakiet seaborn, wykonując polecenie w powłoce:

```
$ sudo /opt/anaconda3/bin/conda install seaborn
```

lub w wierszu poleceń:

```
[anaconda3] > conda install seaborn
```

Aby sprawdzić, czy pakiet został zainstalowany w sposób poprawny, możemy wykonać np. następujący ciąg instrukcji.

```
>>> import seaborn as sns
>>> sns.__version__
'0.7.0'
```

W ten posób przy okazji poznaliśmy numer wersji tego pakietu.

CIEKAWOSTKA

Jeśli pod systemem Linux nie mamy wystarczających uprawnień do instalacji nowych pakietów (np. na komputerach w firmie lub laboratorium, gdzie administratorami są inne osoby), możemy wykonać następujące polecenia:

```
$ cd ~                                  # przejdź do katalogu domowego
$ /opt/anaconda3/bin/conda create -n my_root --clone=/opt/anaconda3
$ ~/.conda/envs/my_root/bin/python      # tu jest teraz interpreter
$ ~/.conda/envs/my_root/bin/conda install seaborn       # instalacja
$ ~/.conda/envs/my_root/bin/conda update --all          # aktualizacja
```

WAŻNE

Zwróćmy także uwagę, że Anaconda dystrybuuje własne wersje tylko niektórych pakietów z PyPI; zob. docs.continuum.io/anaconda/pkg-docs/. Gdy potrzebny nam pakiet nie jest dostępny za pośrednictwem programu conda (np. youtube-dl), możemy skorzystać z programu pip.

1.2. Notatniki Jupyter

Na dłuższą metę praca ze standardowym interpreterem nie jest zbyt wygodna, zwłaszcza w interesującym nas obszarze zastosowań. Z tego względu sugerujemy użycie nakładki *Jupyter* (znanej wcześniej pod nazwą IPython Notebook, od ang. *Interactive Python*). Umożliwia ona tworzenie interaktywnych *notatników*, które mogą jednocześnie grać rolę raportów z analizy danych.

W każdym z notatników mogą znajdować się m.in.:
— wstawki z kodem języka Python;
— wyniki obliczeń (takie, jakie są generowane na konsoli);
— tabele;
— wykresy.

Najpierw skupimy się na samych obliczeniach. Sposoby formatowania tekstu z wykorzystaniem języka Markdown omówimy nieco dalej.

ZADANIE 1.4. Uruchom serwer Jupyter, wywołując:

```
$ /opt/anaconda3/bin/jupyter-notebook
```

lub klikając *Menu Start* → *Anaconda3* → *Jupyter Notebook* pod Windows.

Jeśli program jupyter-notebook uruchamiamy z poziomu powłoki, warto wcześniej zmienić katalog roboczy na ten, w którym chcemy przechowywać notatniki. Dokonujemy tego przy użyciu polecenia cd.

Zwróćmy uwagę, że Jupyter automatycznie otwiera okno przeglądarki internetowej. To właśnie za jej pośrednictwem tworzymy nowy notatnik, klikając *New* → *Python 3 Notebook*. Warto od razu zmienić jego nazwę (domyślna to *Untitled*). Pliki notatnika są zapisywane w katalogu roboczym z rozszerzeniem `.ipynb`.

ZADANIE 1.5. Zapoznaj się ze strukturą menu w oknie notatnika. Zwróć uwagę m.in. na opcję *Kernel* → *Interrupt*, która pozwala przerywać zbyt długo wykonujące się obliczenia.

Aby zakończyć działanie serwera Jupyter, należy dwukrotnie wcisnąć klawisze ⟨CTRL+C⟩ w powłoce lub wierszu poleceń.

1.2.1. Tryby pracy

Praca z notatnikami Jupyter może się odbywać w dwóch trybach, a dokładniej:
— w trybie edycji (wprowadzanie kodu, ang. *edit mode*);
— w trybie poleceń (zarządzanie komórkami, ang. *command mode*).

WAŻNE

Tryb edycji kodu w komórce roboczej włączamy, używając klawisza ⟨ENTER⟩. Do *trybu poleceń* przechodzimy z kolei, wciskając klawisz ⟨ESC⟩.

Pracę z notatnikiem rozpocznijmy od wypisania przykładowego napisu.

ZADANIE 1.6. Wprowadź w komórce notatnika polecenie `print("Dzień dobry.")`.

Ewaluacja kodu następuje po naciśnięciu np. kombinacji klawiszy ⟨CTRL+ENTER⟩ lub ⟨ALT+ENTER⟩. Pierwsza z nich powoduje natychmiastowe przejście do trybu poleceń, a druga – utworzenie nowej komórki w trybie edycji.

```
>>> print("Dzień dobry.")
Dzień dobry.
```

Na potrzeby niniejszej książki wprowadzane instrukcje najczęściej będziemy wyróżniać przy użyciu „>>>”. Dla większej czytelności znaki te będziemy jednak pomijać w przypadku dłuższych fragmentów kodu.

WAŻNE

Język Python rozróżnia wielkość liter (ang. *case sensitive*). Wywołanie `PRINT("...")` zakończy się więc błędem.

Powyższy kod jest oczywiście niezmiernie prosty i nie wymaga właściwie żadnego tłumaczenia. W niedalekiej przyszłości zaczniemy jednak pisać bardziej złożone (pod)programy, np. własne funkcje (rozdz. 6), klasy (rozdz. 16) czy projekty składające się z wielu modułów (rozdz. 15). Jest więc ważne, by kod, który tworzymy, był przejrzysty i czytelny. W tym celu przydadzą się nam *komentarze*. Tworzymy je przy użyciu znaku „#”, np.:

```
>>> # Dlaczego powiedział pan „dzień dobry” skoro jest już popołudnie?
```

Komentarze są całkowicie ignorowane przez interpreter języka Python. Jako komentarz są tu traktowane wszystkie znaki od znaku „#” aż do końca wiersza:

```
>>> print("Jesteśmy gotowi.") # Wchodzą osoby w syberyjskich futrach.
Jesteśmy gotowi.
```

Dostęp do *systemu pomocy* zapewnia funkcja `help()`, która przyjmuje jako argument nazwę interesującej nas funkcji lub obiektu. Na przykład, wykonanie instrukcji:

```
>>> help(type)
```

zwróci informacje na temat funkcji `type()`, która pozwala sprawdzić typ danego obiektu. Można też skorzystać z zapisu `?type` lub `type?`, jest on jednak rozszerzeniem udostępnianym tylko przez Jupyter.

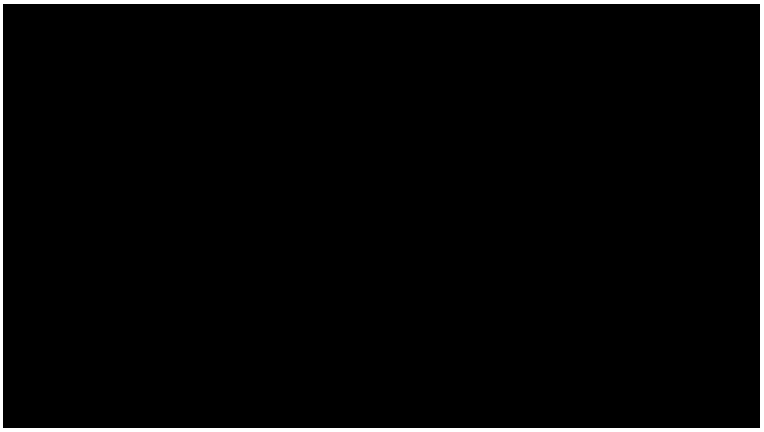
Zwróćmy także uwagę na to, że pisane przez nas polecenia mogą służyć do generowania wykresów. Jeśli chcemy, by Jupyter automatycznie umieszczał je w notatniku, wykonujemy specjalną dyrektywę:

```
>>> %matplotlib inline
```

Wprowadźmy, co następuje.

```
import matplotlib.pyplot as plt    # ładujemy potrzebne pakiety
import numpy as np                 # "as" – alias dla nazwy pakietu
x = np.linspace(0, 4*np.pi, 100)   # generujemy dane
y = np.sin(x)
plt.plot(x, y)                     # rysujemy
plt.show()
```

Wynikowy wykres zamieszczamy na rys. 1.1.



Rysunek 1.1. Przykładowy wykres; zob. powyższy kod

CIEKAWOSTKA

Przy użyciu „%” wprowadzamy specjalne dyrektywy (ang. *IPython magics*), które są rozszerzeniem udostępnianym przez serwer Jupyter: nie są częścią języka Python. W szczególności dyrektywy te nie zadziałają w zwykłym skrypcie języka Python uruchamianym z poziomu powłoki. Wśród nich mamy np. `%timeit`, który służy do pomiaru czasu wykonania danej instrukcji, np.:

```
>>> %timeit -n 1000 2+2          # zmierz czas wykonania „2+2”
```

Powyżej dyrektywę tę wywołaliśmy z dodatkowym argumentem `-n 1000`, który określa, ile razy ma zostać wykonane podane dodawanie dwóch liczb całkowitych.

1.2.2. Najważniejsze skróty klawiszowe

Aby ułatwić sobie pracę z notatnikami, warto zapoznać się z podstawowymi skrótami klawiszowymi.

ZADANIE 1.7. Wciśnij ⟨H⟩ w trybie poleceń i przeczytaj opis dostępnych skrótów klawiszowych.

W tabeli 1.1 przedstawiamy skróty używane przez nas najczęściej w trybie poleceń. Umożliwiają one np. na tworzenie, uruchamianie, edycję oraz usuwanie nowych komórek.

Tabela 1.1. Podstawowe skróty klawiszowe w trybie poleceń

Skrót klawiszowy	Znaczenie
⟨CTRL + ENTER⟩	wykonanie instrukcji zawartych w komórce
⟨ALT + ENTER⟩	wykonanie instrukcji i utworzenie nowej komórki poniżej
⟨SHIFT + ENTER⟩	wykonanie instrukcji i przeniesienie kursora do następnej komórki
⟨B⟩	utworzenie nowej pustej komórki pod komórką roboczą (tą, w której stoi kursor)
⟨A⟩	utworzenie nowej pustej komórki nad komórką roboczą
⟨DD⟩	usunięcie komórki roboczej
⟨X⟩, ⟨C⟩, ⟨V⟩	odpowiednio: wycinanie, kopiowanie, wklejanie komórki
⟨Z⟩	wycofanie wprowadzonych zmian
⟨H⟩	wyświetlenie listy skrótów klawiszowych
⟨Y⟩	oznaczenie, że w komórce znajduje się kod w języku Python przeznaczony do ewaluacji (zachowanie domyślne)
⟨M⟩	oznaczenie, że w komórce znajduje się kod w języku Markdown (zob. p. 1.2.3)

Z kolei w trybie edycji mamy dostęp m.in. do automatycznego uzupełniania wprowadzanych nazw (⟨TAB⟩), wyróżniania tekstu wcięciem (⟨CTRL+]⟩, ⟨CTRL+[⟩), umieszczania bloku tekstu w komentarzu (⟨CTRL+/⟩), a także wszystkich skrótów klawiszowych znanych z innych edytorów tekstowych.

1.2.3. Podstawy języka Markdown

Skróty klawiszowe ⟨Y⟩ oraz ⟨M⟩ (w trybie poleceń) pozwalają zmieniać typ komórki roboczej. Domyślnie są tworzone komórki, w których po prostu wprowadzamy instrukcje języka Python. W przypadku tworzenia raportów z analizy danych przydatna jest jednak także możliwość wprowadzania „wolnych" partii tekstu oraz jego formatowania. Do tego celu możemy wykorzystać Markdown, który jest bardzo popularnym i prostym językiem znaczników.

WAŻNE

Wynik działania poleceń w języku Markdown możemy zobaczyć, wciskając klawisze ⟨CTRL+ENTER⟩, ⟨ALT+ENTER⟩ lub ⟨SHIFT+ENTER⟩.

Podział tekstu na akapity, sekcje itd. Aby wprowadzony tekst podzielić na akapity, należy w stosownym niejscu wstawić dwa puste wiersze.

Poszczególne części tekstu możemy pogrupować, wstawiając nagłówki. Tworzymy je przy użyciu ciągu znaków „#”. Rozmiar nagłówka zależy od liczby znaków „#” umieszczonych przed stosowną etykietą. W szczególności mamy:

```
#  Nagłówek 1
## Nagłówek 2

Przepraszam, mam katar.
Przyszedłem      z       reklamacją.

Przykro mi, ale teraz mam przerwę
                           obiadową.
```

Nagłówek 1

Nagłówek 2

Przepraszam, mam katar. Przyszedłem z reklamacją.

Przykro mi, ale teraz mam przerwę obiadową.

Formatowanie tekstu. Fragmenty tekstu możemy wyróżnić np. przy użyciu pogrubienia lub kursywy. Odpowiednie style uzyskujemy, korzystając z następującej notacji:

```
*kursywa*
**pogrubienie**
`kod`
~~przekreślenie~~
```

kursywa
pogrubienie
`kod`
~~przekreślenie~~

Wypunktowania. Listy wypunktowane tworzymy, stawiając znak „*” przed każdym elementem, np.:

```
* David
* Na pewno
```

- David
- Na pewno

Z kolei listy o elementach numerowanych tworzymy przy użyciu liczb i kropki, np.:

```
1. David
2. Thomas
```

1) David
2) Thomas

Wprowadzanie kodu. Kod, który chcemy jedynie zacytować, możemy wprowadzić w następujący sposób:

````
```python
x = 7
print(x)
```
````

```
x = 7
print(x)
```

Wzory i wyrażenia matematyczne. Wzory oraz wyrażenia matematyczne możemy umieszczać w notatkach, korzystając z konwencji języka TEX; por. [29, 47]:

```
Wzór *inline*: $f(x)$

Tryb blokowy:
$$f(x) = \frac{\sqrt{x}}{2}$$
```

Wzór *inline*: $f(x)$
Tryb blokowy:

$$f(x) = \frac{\sqrt{x}}{2}$$

Tabele. Markdown umożliwia także bardzo łatwe wprowadzanie tabel. Dokonujemy tego przy użyciu znaków „-” oraz „|”, jak w poniższym przykładzie:

```
| Lp. | Danie    |
|-----|----------|
| 1.  | Mielonka |
| 2.  | Mielonka |
```

Lp.	Danie
1.	Mielonka
2.	Mielonka

Odnośniki. W dokumentach tworzonych przy użyciu języka Markdown możemy zamieszczać odnośniki do stron internetowych. W tym celu korzystamy z nawiasów kwadratowych, w których umieszczamy etykietę do wyświetlenia w tekście. Sam adres docelowy podajemy w nawiasach okrągłych. Na przykład:

```
[kliknij mnie](http://python.org)
```

kliknij mnie

Obrazki. W podobny sposób możemy wstawiać obrazki. Tym razem jednak poprzedzamy całe polecenie wykrzyknikiem, a w nawiasach okrągłych umieszczamy odnośnik lub ścieżkę dostępu do pliku na dysku.

```
![etykietka](https://www.python.org/static/img/python-logo.png)
```

ZADANIE 1.8. W notatniku Jupyter utwórz kurs podstaw języka Markdown (podobny do niniejszego) przy użyciu poleceń tego języka oraz wyeksportuj go do pliku w formacie HTML.

1.3. Pierwsze kroki w języku Python

W rozdziałach 2–6 przedstawimy w wyczerpujący sposób podstawy języka Python, bez których trudno sobie wyobrazić dokonywanie bardziej zaawansowanych obliczeń. Abyśmy mogli szybciej przyzwyczaić się do składni najbardziej kluczowych poleceń, dokonajmy już w tym miejscu ich krótkiego przeglądu.

Typy skalarne. Typy skalarne, do których zaliczamy m.in. liczby całkowite (np. `1`, `-123`), liczby zmiennopozycyjne (np. `3.14`, `5e-16` $= 5 \cdot 10^{-16}$) oraz stałe logiczne (`True`, `False`) omówimy w rozdz. 2.

Zmienne. Zmienne, które mogą przechowywać dowolne wartości (obiekty) i do których możemy odwoływać się przy użyciu czytelnej nazwy, tworzymy przy użyciu

operatora przypisania, „=” (zob. p. 2.1.3). Poniżej tworzymy np. zmienną o nazwie x i przypisujemy do niej wartość całkowitą 2, a następnie używamy jej do wykonania prostego działania arytmetycznego.

```
>>> x = 2
>>> print(x)                    # co się stanie, gdy w notatniku pominiemy print()?
2
>>> x * 2 + 1
5
```

Napisy. Napisy, z których korzystaliśmy już powyżej, omówimy szczegółowo zarówno w rozdz. 2, jak i 3.

```
>>> "mielonka"
'mielonka'
>>> 'szynka i mielonka'       # tak też można
'szynka i mielonka'
>>> """
... mielonka
... mielonka
... szynka i mielonka
... i inne przysmaki
... """
...
'\nmielonka\nmielonka\nszynka i mielonka\ni inne przysmaki\n'
```

Odnotujmy w tym miejscu, że znak „\n” oznacza nakaz przejścia do nowego wiersza.

Listy, krotki i zakresy. Obiekty typu sekwencyjnego, takie jak listy:

```
>>> lista = [5, 4, 3, 2, 1]
>>> print(lista)
[5, 4, 3, 2, 1]
```

i krotki:

```
>>> krotka = 5, 4, 3, 2, 1      # równoważnie: (5, 4, 3, 2, 1)
>>> print(krotka)
(5, 4, 3, 2, 1)
```

oraz ich własności omówimy w rozdz. 3. Służą one do tworzenia ciągów obiektów dowolnego typu. Pierwszy z nich należy do grupy typów *zmienialnych* (ang. *mutable*), czyli takich, których elementy można dowolnie modyfikować.

```
>>> lista[0] = 6       # zmień pierwszy element
>>> print(lista)
[6, 4, 3, 2, 1]
>>> krotka[-1] = -1    # próba modyfikacji ostatniego elementu od końca
```

```
Traceback (most recent call last):
TypeError: 'tuple' object does not support item assignment
```

Co więcej, do typów sekwencyjnych zaliczamy także niezmienialne obiekty klasy zakres, np. `range(10)` reprezentuje ciąg liczb całkowitych 0, 1, . . . , 9.

Słowniki i zbiory. W rozdziale 4 omówimy słowniki, czyli kolekcje elementów postaci „`klucz`: `wartość`", np.:

```
>>> {
...     1: "David",
...     2: "Thomas"
... }
...
{1: 'David', 2: 'Thomas'}
```

Każda z `wartości` jest identyfikowana przy użyciu odpowiadającego mu jednoznacznego `klucza`. Rozważymy tam również obiekty typu zbiór – składają się one z unikatowych elementów, które nie są wymienione w żadnej szczególnej kolejności.

```
>>> { "David", "Na pewno", "Thomas" }
{'Thomas', 'Na pewno', 'David'}
```

Instrukcja warunkowa. Instrukcja `if`, zob. podrozdz. 5.1, służy do warunkowego wykonywania podanego bloku kodu. I tak w poniższym przykładzie sprawdzamy, czy zmienna `x` ma wartość mniejszą niż 4. Jeśli tak jest w istocie, wypisujemy uśmieszek.

```
>>> x = 2
>>> if x < 4:          # uwaga na dwukropek
...     print(":)")    # wyróżnienie przy użyciu 4 spacji
...
:)
```

Gdy podany warunek logiczny nie jest spełniony, możemy także wykonać jakąś alternatywną instrukcję. Służy do tego klauzula `else`. W poniższym przykładzie, jeśli zmienna `x` nie jest mniejsza od 4, jesteśmy po trzykroć smutni.

```
>>> x = 8 * x
>>> if x < 4:
...     print(':)')   # 4 spacje są naprawdę zalecane
... else:
...     print(':(')   # 4 spacje to złota reguła
...     print(':(')   # tutaj też nie będziemy zbyt oryginalni
...     print(':(')   # ani tu
...
:(
:(
:(
```

ZADANIE 1.9. Język Python wymusza spójne formatowanie kodu. Spróbuj w powyższym przykładzie poeksperymentować z liczbą spacji, przy użyciu których tworzymy bloki instrukcji do warunkowego wykonania. Zobacz, w jakich przypadkach niekonsekwencja w tym zakresie doprowadzi do wystąpienia błędu.

Pętle. Pętle (zob. podrozdz. 5.2), służą do wykonywania tych samych instrukcji wielokrotnie, ale być może na różnych danych wejściowych. W pętli `for` fragment kodu jest wykonywany na kolejnych elementach obiektu iterowalnego (np. listy, krotki i zakresu) podanego jako argument. W poniższym przykładzie rozpatrujemy wartości zmiennej `i`, kolejno, 0, 1, 2 i 3.

```
>>> for i in range(4):          # liczby całkowite z przedziału [0, 4)
...     print(i**2, end=", ")   # potęgowanie
...
0, 1, 4, 9,
```

Z kolei instrukcje wewnątrz pętli `while` są wykonywane dopóty, dopóki jest spełniony pewien warunek logiczny. Poniżej wielokrotnie odejmujemy liczbę 1 od zmiennej `x`, aż ta zmienna osiągnie wartość mniejszą lub równą 0.

```
>>> x = 5
>>> while x > 0:
...     print(x, end=", ")
...     x -= 1                  # równoważnie: x = x - 1
...
5, 4, 3, 2, 1,
```

Zgłaszanie wyjątków. Jeśli chcemy wskazać, że jakaś czynność zakończyła się niepowodzeniem, możemy zgłosić wyjątek:

```
>>> raise Exception("Nikt nie przejdzie.")
Traceback (most recent call last):
Exception: Nikt nie przejdzie.
```

W podrozdziale 5.3 dowiemy się, w jaki sposób obsługiwać tego rodzaju sytuacje.

Funkcje. Funkcję (zob. rozdz. 6) tworzymy przy użyciu słowa kluczowego `def`, po którym wpisujemy jej nazwę oraz ujętą w nawiasy okrągłe listę parametrów. Po znaku „:” podajemy blok poleceń, które ma ona wykonać na podanych jej argumentach, czyli tzw. ciało funkcji, np.:

```
>>> def kwadrat(x):
...     return x**2             # natychmiast zwróć podaną wartość
...
>>> kwadrat(3)
9
```

2 TYPY SKALARNE

Zanim poznamy zaawansowane aspekty programowania w języku Python, a w szczególności ich zastosowanie w przetwarzaniu i analizie informacji, powinniśmy dokonać przeglądu podstawowych typów danych. W niniejszym rozdziale przyjrzymy się bliżej *typom skalarnym*:

— wartościom liczbowym;
— wartościom logicznym;
— napisom[1].

2.1. Liczby

Wyróżniamy trzy podstawowe typy liczbowe:

— całkowite (ang. *integers*);
— zmiennopozycyjne (ang. *floating-point numbers*);
— zespolone (z ang. *complex numbers*).

Liczby całkowite. Rozpocznijmy nasze rozważania od *liczb całkowitych*. Ich zbiór, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$, oznaczamy w matematyce najczęściej jako $\mathbb{Z}$. Stałe całkowite w języku Python przeważnie wprowadzamy, podając po prostu ciąg cyfr (bez kropki dziesiętnej), ewentualnie poprzedzając go znakiem „-” lub „+”.

```
>>> -54321
-54321
```

Aby sprawdzić, jakiego typu (jakiej klasy; por. rozdz. 16) jest powyższy obiekt, możemy użyć wbudowanej funkcji `type()`. Będziemy z niej korzystać wielokrotnie, poznając nowe i coraz bardziej złożone typy danych. Wykonując więc:

```
>>> type(-54321)
<class 'int'>
```

[1] W rozdziale 3 okaże się, że każdy napis możemy postrzegać również jako reprezentanta typu sekwencyjnego.

otrzymaliśmy informację, że liczba `-54321` jest typu `int` (od ang. *integer*).

CIEKAWOSTKA

Wartości całkowite są reprezentowane przy użyciu 32 lub 64 bitów (por. `sys.maxsize`). Jednak w przypadku przekroczenia ich zakresu Python automatycznie „przełącza" się na tzw. reprezentację *bigint*.

```
>>> sys.maxsize       # w innych językach – ograniczenie górne
9223372036854775807
>>> print(2 ** 128)   # operator „**" oznacza potęgowanie
340282366920938463463374607431768211456
>>> type(2 ** 128)    # w pamięci komputera: bigint
<class 'int'>
```

Zmiana ta odbywa się w sposób niejawny – nas, jako użytkowników, nie powinna więc zbytnio interesować.

CIEKAWOSTKA

Możemy także wprowadzać wartości całkowite w systemach liczbowych innych niż dziesiętny. W tym celu korzystamy z przedrostków `0b`, `0o` i `0x`, odpowiednio, dla liczb w systemie dwójkowym, ósemkowym i szesnastkowym. Na przykład:

— notacja dwójkowa (binarna):

```
>>> print(0b011) # 0*2**2 + 1*2**1 + 1*2**0
3
```

— notacja ósemkowa (oktalna, używana np. przy podawaniu praw dostępu do plików):

```
>>> print(0o123) # 1*8**2 + 2*8**1 + 3*8**0
83
```

— notacja szesnastkowa (heksadecymalna, używana np. podczas określania barw):

```
>>> print(0x9f)  # 9*16**1 + 15*16**0
159
```

Liczby zmiennopozycyjne. *Liczby zmiennopozycyjne* stanowią podzbiór liczb rzeczywistych, $\mathbb{R}$. W odróżnieniu od wartości typu `int` wprowadzamy je, podając m.in. jawnie część ułamkową (przy użyciu kropki):

```
>>> -543231.0
-543231.0
>>> type(-543231.0)
<class 'float'>
```

Możemy skorzystać także z tzw. *notacji naukowej*:

```
>>> 5.4321e4  # „razy 10 do potęgi 4"
54321.0
>>> -1e0
-1.0
```

WAŻNE

Wartości `1` i `1.0` nie są tego samego typu, chociaż z matematycznego punktu widzenia reprezentują tę samą liczbę.

Liczby zespolone. Część urojoną *liczby zespolonej* wprowadzamy przy użyciu przyrostka `j` lub `J`.

```
>>> 1 + 2j
(1+2j)
>>> type(1.0 + 2.0J) # ta sama liczba
<class 'complex'>
```

Mimo że liczb zespolonych nie stosuje się zbyt często w analizie danych, przyrostek `j` warto zapamiętać – np. pakiet numpy wykorzystuje go do generowania ciągów; por. s. 107.

2.1.1. Operatory arytmetyczne

Dokonajmy przeglądu podstawowych operatorów arytmetycznych, które możemy stosować na poznanych wartościach skalarnych.

Dodawanie, odejmowanie, mnożenie, dzielenie. Przede wszystkim mamy dostęp do binarnych (dwuargumentowych) operatorów „+" (dodawanie), „−" (odejmowanie), „*" (możenie) i „/" (dzielenie):

```
>>> 1   + 2          # dodawanie
3
>>> 7.0 - 1.3        # odejmowanie
5.7
>>> 4   * 3.5        # mnożenie
14.0
>>> 1   / 2          # dzielenie
0.5
```

Zwróćmy uwagę, że w przypadku dzielenia dwóch liczb całkowitych otrzymaliśmy liczbę zmiennopozycyjną.

```
>>> type(1 / 2)
<class 'float'>
```

Co więcej, w przypadku działań na różnych typach nastąpiła automatyczna promocja typu szczegółowego do bardziej ogólnego. Stąd w wyniku działania `4 * 3.5` otrzymaliśmy wartość typu `float`:

```
>>> type(4 * 3.5)
<class 'float'>
```

Odnotujmy także, że mnożenie i dzielenie ma wyższy priorytet niż dodawanie i odejmowanie, co jest całkowicie zgodne z naszą matematyczną intuicją:

```
>>> 1 + 2 * 3       # to samo, co 1 + (2 * 3)
7
```

Przy użyciu nawiasów możemy jednak wymusić inną kolejność działań:

```
>>> (1 + 2) * 3
9
```

Dzielenie całkowite i reszta z dzielenia. Bardziej zaawansowane działania na liczbach uwzględniają m.in. powiązane ze sobą dzielenie całkowitoliczbowe oraz resztę z takiego dzielenia (tzw. dzielenie *modulo*). Dla danych `x` i `y` operacje te definiujemy jako:

$$\frac{\mathtt{x}}{\mathtt{y}} = \mathtt{q} \text{ z resztą } \mathtt{r},$$

gdzie `q` jest wynikiem dzielenia całkowitoliczbowego, a `r` – operacji *modulo*. Pierwsze z nich realizuje operator binarny „`//`”, a drugie – „`%`”. Zachodzi więc `x = q*y+r`. Na przykład:

```
>>> 7 // 3
2
>>> 7 % 3
1
```

Co ciekawe, dzielenie tego typu możemy wykonać także na liczbach zmiennopozycyjnych, w tym ujemnych. Bardziej formalnie, w powyższym wzorze mamy $\mathtt{q} = \lfloor \mathtt{x}/\mathtt{y} \rfloor$, gdzie $\lfloor u \rfloor = \max\{v \in \mathbb{Z} : v \leqslant u\}$ oznacza funkcję *podłoga*, tj. największą liczbę całkowitą nie większą niż dany argument. Ponadto `r` jest zawsze takie, że $\mathtt{r} \in [0, \mathtt{y})$ albo $\mathtt{r} \in (\mathtt{y}, 0]$ (w zależności od znaku `y`).

```
>>> 2.3 // 1.2
1.0
>>> 2.3 % 1.2
1.0999999999999999
>>> -13 // -1.5
8.0
>>> -13 % -1.5
-1.0
```

CIEKAWOSTKA

W niektórych językach programowania (np. w C) znak wartości *modulo* może nie być zgodny ze znakiem dzielnika. W języku Python jest jednak inaczej:

```
>>> -3%3, -2%3, -1%3, 0%3, 1%3, 2%3, 3%3, 4%3
(0, 1, 2, 0, 1, 2, 0, 1)
```

Odnotujmy, że wyniki powyższych działań są zapętlone (0, 1, 2, 0, 1, 2, . . .). Takie zachowanie może się przydać m.in. podczas indeksowania tablic lub operowania na datach. Na przykład załóżmy, że mamy ciąg składający się z 7 elementów, gdzie każdy z nich reprezentuje inny dzień tygodnia (0 – poniedziałek, . . . , 6 – niedziela) i chcemy odwołać się do dnia, który jest 2 dni przed wtorkiem:

```
>>> (1 - 2) % 7 # wtorek minus 2 dni = niedziela
6
```

Potęgowanie. Kolejną ważną operacją jest potęgowanie. Realizuje je operator „**”, np.:

```
>>> 2 ** 4
16
```

Warto zwrócić uwagę na fakt, że ma ono wyższy priorytet nie tylko niż dodawanie i mnożenie, ale i operacja zmiany znaku (unarny minus). Stąd:

```
>>> -2 ** 4          # równoważnie: -(2 ** 4)
-16
>>> (-2) ** 4
16
>>> 1 + 2 * 3 ** 4 # 1 + (2 * (3 ** 4))
163
```

Pamiętajmy, by w razie wątpliwości co do kolejności wykonywania działań, zwłaszcza w przypadku skomplikowanych operacji arytmetycznych, zawsze stosować nawiasy.

Działania na liczbach zespolonych. W przypadku liczb zespolonych możemy wykonać część operacji arytmetycznych, takich jak dodawanie lub odejmowanie:

```
>>> (1+2j) + (2+1j)
(3+3j)
```

mnożenie:

```
>>> (1+2j) * (2+1j)
5j
>>> (1+2j) * (2-1j)
(4+3j)
```

i dzielenie:

```
>>> (2+2j) / 2
(1+1j)
>>> (2+2j) / 2j
(1-1j)
>>> (2+2j) / (1+2j)
(1.2-0.4j)
```

2.1.2. Konwersja typów

Przy użyciu funkcji[2] `int()`, `float()` oraz `complex()` możemy dokonać jawnej konwersji obiektu jednego typu na inny. Na przykład, wywołanie `int()` na liczbie zmiennopozycyjnej prowadzi do obcięcia jej części ułamkowej:

```
>>> int(3.99)
3
>>> int(-3.99)
-3
```

Dalej:

```
>>> int("9f", 16)      # liczba szesnastkowa
159
>>> float(1)
1.0
>>> complex(1)
(1+0j)
>>> complex(1.7, 6)
(1.7+6j)
```

[2] A właściwie konstruktorów klas `int`, `float` i `complex`. Konstruktory (por. rozdz. 16) służą do tworzenia nowych obiektów określonego typu – takiego jak ich nazwa.

2.1.3. Tworzenie obiektów nazwanych

Obiekty nazwane, czyli *zmienne*, tworzymy przy użyciu operatora przypisania, „=”. Jako proste ćwiczenie utwórzmy zmienną o nazwie x i przypiszmy do niej wartość całkowitą.

```
>>> x = 2
>>> print(x)
2
>>> print(x * 3 + 1)  # użycie w obliczeniach
7
>>> type(x)
<class 'int'>
```

Python jest językiem *dynamicznie typowanym*, więc tworzonej zmiennej nie trzeba w żaden sposób wcześniej deklarować. Co więcej, raz utworzona zmienna może zostać potem zastąpiona inną – zarówno pod względem typu, jak i wartości.

```
>>> x = 3.14
>>> print(x)
3.14
>>> type(x)
<class 'float'>
```

CIEKAWOSTKA

W powyższym przykładzie nadpisaliśmy wartość istniejącej zmiennej. Dzięki wbudowanemu mechanizmowi *odśmiecania pamięci* (ang. *garbage collection*) środowisko Python samoczynnie wykrywa, że poprzednia wartość (2) nie jest już przypisana do żadnej innej zmiennej – może więc zostać usunięta z pamięci. O ile w przypadku pojedynczej wartości skalarnej taki automatycznie uruchamiany proces nie jest zbyt efektowny, o tyle w przypadku operowania na dużych zbiorach danych, jego działanie jest nieocenione.

ZADANIE 2.1. Niech dane będą dwie zmienne, np.:

```
>>> a = 1
>>> b = 2
```

Zamień ich wartości miejscami.

Rozwiązanie. Klasyczne podejście polega na utworzeniu zmiennej tymczasowej, w której będziemy przechowywać jedną z przestawianych wartości, np.:

```
>>> c = a
>>> a = b
>>> b = c
```