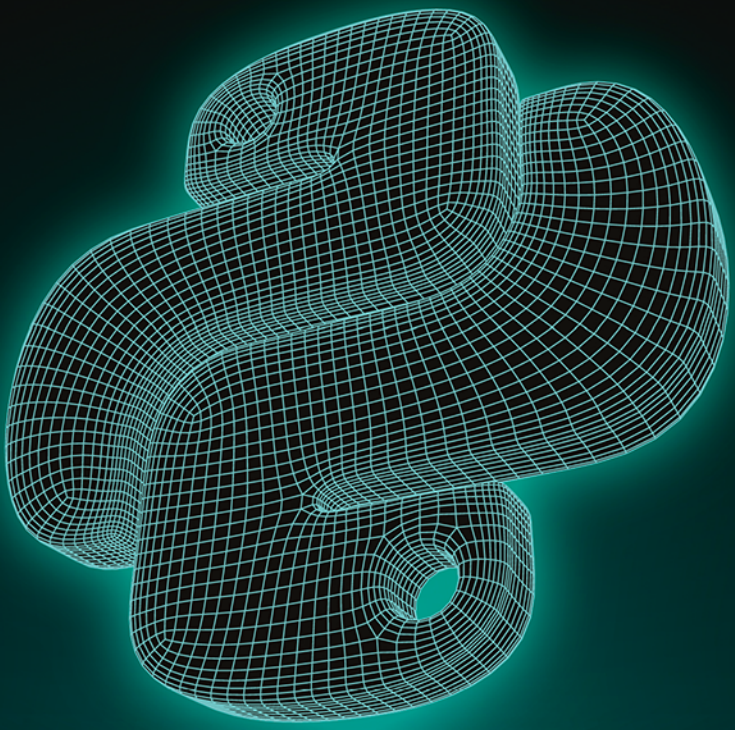
Adam Jurkiewicz

Python 3

Projekty dla początkujących i pasjonatów



Helion EDUKACJA

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Materiały graficzne w książce i na okładce zostały wykorzystane
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/pytmiv
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-289-0602-0

Printed in Poland.

SPIS TREŚCI

WSTĘP **5**

1 PODSTAWY JĘZYKA PYTHON **10**
1.1. Po co w ogóle ten Python? 11
1.2. Podstawowa wiedza o Pythonie 21
1.3. Pętle 37
1.4. Funkcje 53
1.5. Importowanie modułów 59
1.6. Klasy obiektowe i podstawowe operacje niezbędne do tworzenia nowych klas i metod 66
1.7. Przestrzenie nazw — dostęp do różnych obiektów 79
Podsumowanie 83

2 GRY WYKORZYSTUJĄCE MECHANIKĘ RUCHU — PONG **85**
2.1. Czym jest Pong? 85
2.2. Od czego zacząć: moduł Pygame Zero 86
2.3. Wprawiamy piłeczkę w ruch 93
2.4. Paletki wchodzą do gry 96
2.5. Programujemy wygraną 105
Pełny kod 106

3 GRA W POŁĄCZENIU Z FIZYKĄ **110**
3.1. Rzut poziomy 110
3.2. Wykres 112
Pełny kod 123

4 SYMULACJE FIZYCZNE — ANIMACJA UKŁADU SŁONECZNEGO 126
4.1. Symulacja Układu Słonecznego 127
4.2. Ruch planet w Układzie Słonecznym — podstawy teoretyczne 133
4.3. Trudny kod — plik r4_04.py 142
Pełny kod 149

5 PREZENTACJA I WIZUALIZACJA DANYCH 155
5.1. Moduł pandas 156
5.2. Wykres 161
Pełny kod 162

6 WIZUALIZACJA DANYCH NA MAPIE 167
6.1. Moduł Cartopy 168
6.2. Przygotowanie i edycja danych 171
6.3. Przygotowanie mapy 177
Pełny kod 180

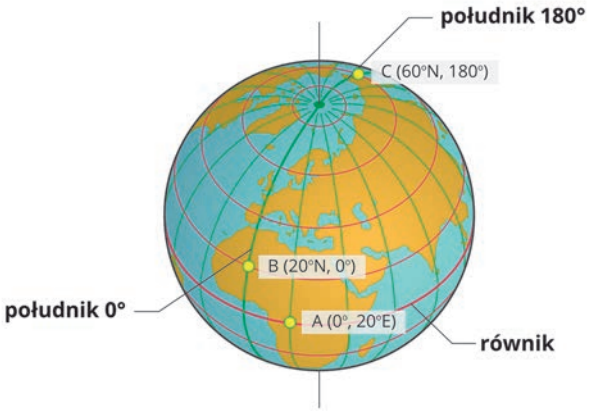
7 ANONIMIZOWANIE DANYCH EXIF W ZDJĘCIACH 182
7.1. Odczytywanie informacji o zdjęciach 182
7.2. Anonimizowanie danych o zdjęciach 186
7.3. Rekurencja — czyli wykonywanie funkcji przez nią samą 194
7.4. Sprawdzanie rekurencyjne katalogów w języku Python 195
Pełny kod 204

8 SYSTEM OPERACYJNY MA ZNACZENIE 207
8.1. Komputer 207
8.2. System operacyjny 208
8.3. Sprzęt i systemy operacyjne używane w pracy nad książką 213
8.4. Środowisko testowe 215
8.5. Legalność oprogramowania 223
8.6. Przykład konfiguracji systemu do codziennej pracy 225

9 KONIEC… A WŁAŚCIWIE POCZĄTEK 227

ŹRÓDŁA 229

WIZUALIZACJA DANYCH NA MAPIE



Rysunek 6.1. Wizualizacja kuli ziemskiej

Czy w czasach, kiedy prawie każda osoba ma w kieszeni urządzenie z nawigacją satelitarną, znajomość podstawowych zasad dotyczących geografii i map jest potrzebna? Tego nie wiemy dokładnie, ale pewne podstawy zawsze warto mieć w pamięci, choćby na wypadek awarii tych urządzeń. Np. długość i szerokość geograficzna — co oznaczają, jak je wykorzystać? W dawnych czasach taka wiedza była bardzo przydatna, dziś traci na znaczeniu w codziennym życiu, ale wciąż warto znać pewne podstawy. Więcej informacji o długości i szerokości geograficznej można znaleźć na stronach serwisu Epodręczniki.pl.

Kod QR prowadzący do serwisu Epodręczniki.pl, do materiału o długości i szerokości geograficznej — https://epoedreczniki.pl/a/wspolrzedne--geograficzne/D19UAs8Ag

Spróbujemy wykorzystać takie dane do wyświetlenia punktów na mapie. Skorzystamy z modułów `matplotlib` i `cartopy`. Jeden już znasz, a drugi poznasz w tym rozdziale. Dodatkowo nauczysz się czytać dane z plików tekstowych `CSV` — to bardzo przydatna umiejętność w kontekście matury z informatyki; często zdarzają się na niej zadania, w których trzeba wczytać i wyodrębnić pewne dane z plików tekstowych.

6.1. Moduł Cartopy

Moduł `cartopy` jest stworzony do przetwarzania i analizowania danych geograficznych oraz do generowania map. Aby postępować zgodnie z prawem i jego licencją, cytujemy tu jego użycie:

```
Cartopy. Met Office. git@github.com:SciTools/cartopy.git. 2015-02-18.
7b2242e.
```

Dokładny zapis licencyjny brzmi:

```
All Cartopy source code, unless explicitly stated, is © British Crown
copyright, 2016 and is licensed under the GNU Lesser General Public License
as published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version. You should find all source
files with the following header:


© British Crown Copyright 2011 - 2016, Met Office
This file is part of cartopy.

Cartopy is free software: you can redistribute it and/or modify it under the
terms of the GNU Lesser General Public License as published by the Free
Software Foundation, either version 3 of the License, or (at your option)
any later version.

Cartopy is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more
details.

You should have received a copy of the GNU Lesser General Public License
along with cartopy. If not, see https://www.gnu.org/licenses/.
```

Nasz program rozpoczniemy tak, jak to robiliśmy już wielokrotnie: od sprawdzenia, czy odpowiednie moduły są zainstalowane.

```
# program r6_00.py
# Sprawdzamy moduł cartopy

from sys import exit

try:
    import cartopy.crs as crs
    import cartopy.feature as cfeature
    print("Moduł cartopy wczytany.")
except:
    print("Zainstaluj: 'pip install cartopy' ")
    exit(0)

try:
    import matplotlib.pyplot as plt
    print("Moduł matplotlib wczytany.")
except:
    print("Zainstaluj: 'pip install matplotlib' ")
    exit(0)
```

Oczywiście najprawdopodobniej za pierwszym razem zobaczymy komunikat o konieczności instalacji `cartopy`. Odpowiednie polecenia wyglądają tak:

```
# Linux
pip3 install cartopy
# Windows
pip install cartopy
```

Uwaga!

Dla systemu Linux (w moim przypadku Linux Mint 20) należy zainstalować dodatkowe pakiety w systemie operacyjnym oraz dodatkowe moduły Pythona:

```
# Pakiety do systemu Linux
sudo apt-get install libproj-dev proj-data proj-bin
libgeos++-dev
# Dodatkowe moduły Pythona
pip3 install cython scipy
```

Po doinstalowaniu tych pakietów i modułu do języka Python, dalsza instalacja przebiega już prawidłowo:

```
# Na końcu instalacja przebiega poprawnie
pip3 install cartopy

Collecting cartopy
  Using cached Cartopy-0.18.0.tar.gz (14.4 MB)
Requirement already satisfied: numpy>=1.10 in
/usr/local/lib/python3.8/dist-packages (from cartopy)
(1.19.1)
```

```
Collecting pyshp>=1.1.4
  Downloading pyshp-2.1.3.tar.gz (219 kB)
     |████████████████████████████████| 219 kB
1.9 MB/s
Requirement already satisfied: setuptools>=0.7.2 in
/usr/lib/python3/dist-packages (from cartopy) (45.2.0)
Collecting shapely>=1.5.6
  Downloading Shapely-1.7.1-cp38-cp38-manylinux1_x86_64.whl
(1.0 MB)
     |████████████████████████████████| 1.0 MB
19.0 MB/s
Requirement already satisfied: six>=1.3.0 in
/usr/lib/python3/dist-packages (from cartopy) (1.14.0)
Building wheels for collected packages: cartopy, pyshp
  Building wheel for cartopy (setup.py) ... done
  Created wheel for cartopy: filename=Cartopy-0.18.0-cp38-
cp38-linux_x86_64.whl size=15706059
sha256=3a3474efca2f625c780e66448765096416b404bdb1689fa8890e
2155b246a3ba
  Stored in directory:
/home/python/.cache/pip/wheels/7c/3c/68/ed800c08e3e6579b632
fdd26becee97c5c5474625f6c97eca6
  Building wheel for pyshp (setup.py) ... done
  Created wheel for pyshp: filename=pyshp-2.1.3-py3-none-
any.whl size=37262
sha256=72c354048cf5ff0b6fbf1bd5d2dfdc7922b2dbb9c1cf101082cf
62be37131174
  Stored in directory:
/home/python/.cache/pip/wheels/1f/1b/b5/54affbefc8a7e2bdf1d
a000fc576b8a1c91338f1f327a04f4c
Successfully built cartopy pyshp
Installing collected packages: pyshp, shapely, cartopy
Successfully installed cartopy-0.18.0 pyshp-2.1.3 shapely-
1.7.1
```

Uwaga

W systemie MS Windows 10 próba instalacji modułu nie przynosi rezultatu. Spędziłem sporo czasu na wyszukiwaniu zależności i tłumaczeniu komunikatów błędów podczas próby instalacji tego modułu. Na stronie z nieoficjalnymi pakietami dla systemu Windows, w chwili pisania tej książki możemy przeczytać informację:

> *Unofficial Windows Binaries for Python Extension Packages by Christoph Gohlke, Laboratory for Fluorescence Dynamics, University of California, Irvine.*
>
> *Updated on 7 March 2021 at 02:05 UTC.*

Kod QR prowadzący do strony opisującej instalację moduły cartopy w systemie Windows: https://scitools.org.uk/cartopy/docs/latest/installing.html. Dodatkowo należy doinstalować też moduł scipy, dostępny pod adresem: https://pypi.org/project/scipy/

6.2. Przygotowanie i edycja danych

W naszej pracy skorzystamy z prostych danych geograficznych, które stworzymy samodzielnie. Będą dotyczyły położenia kilku miast Polski, określanego za pomocą ich długości i szerokości geograficznej:

Nazwa miasta	Szerokość	Długość
Piła	53.150967	16.738228
Szamotuły	52.611656	16.577906
Kłodzko	50.434563	16.661394
Żnin	52.849384	17.719477
Paczków	50.463993	17.006659
Kazimierz Dolny	51.318043	21.954248
Bolesławiec	51.265854	15.565740
Sztutowo	54.326084	19.179071
Kudowa-Zdrój	50.442715	16.242661
Olkusz	50.281315	19.565687
Lubartów	51.459869	22.609461
Wieliczka	49.987061	20.064796
Police	53.552040	14.572061
Przemyśl	49.783863	22.767792
Myszyniec	53.380493	21.349531
Kielce	50.866077	20.628569
Frombork	54.357479	19.680073
Zakopane	49.299030	19.949047
Sopot	54.444092	18.570328

Nazwa miasta	Szerokość	Długość
Krosno	49.690289	21.754990
Pruszków	52.170471	20.811886
Łeba	54.753063	17.557011
Kędzierzyn-Koźle	50.349880	18.226185
Grodziec	52.038197	18.059839

Dane te zapisujemy w pliku *CSV* (*Comma Separated Value*). Takie pliki charakteryzują się tym, że każdy wiersz, podobnie jak w tabelce, zawiera informacje rozdzielone umownym znakiem, najczęściej przecinkiem. Trzeba też zwrócić uwagę na to, że w długościach i szerokościach geograficznych posłużyliśmy się kropką, a nie przecinkiem, jak zapisuje się te wartości w języku polskim. Jest to spowodowane tym, że takie dane z przecinkiem zamiast kropki nie zostałyby poprawnie zinterpretowane przez program — w językach programowania części dziesiętne oddziela się, podobnie jak w języku angielskim, kropką, a nie przecinkiem.

```
Piła,53.150967,16.738228
Szamotuły,52.611656,16.577906
Kłodzko,50.434563,16.661394
Żnin,52.849384,17.719477
Paczków,50.463993,17.006659
Kazimierz Dolny,51.318043,21.954248
Bolesławiec,51.265854,15.565740
Sztutowo,54.326084,19.179071
Kudowa-Zdrój,50.442715,16.242661
Olkusz,50.281315,19.565687
Lubartów,51.459869,22.609461
Wieliczka,49.987061,20.064796
Police,53.552040,14.572061
Przemyśl,49.783863,22.767792
Myszyniec,53.380493,21.349531
Kielce,50.866077,20.628569
Frombork,54.357479,19.680073
Zakopane,49.299030,19.949047
Sopot,54.444092,18.570328
Krosno,49.690289,21.754990
Pruszków,52.170471,20.811886
Łeba,54.753063,17.557011
Kędzierzyn-Koźle,50.349880,18.226185
Grodziec,52.038197,18.059839
```

Uwaga!

W modułach standardowych, które są dostępne od razu z Pythonem, znajduje się moduł `csv`, dedykowany do obsługi takich plików, ale my nie będziemy z niego korzystać. Zainteresowani mogą zerknąć na stronę dokumentacji.

Kod QR prowadzący do dokumentacji modułu csv — https://docs.python.org/3/library/csv.html

Do przetworzenia takiej niewielkiej porcji danych użyjemy prostego, aczkolwiek dobrze działającego sposobu. Za pomocą konstrukcji `with open... as ...` wczytamy dane z pliku:

```
# Wczytane dane będziemy zapisywać jako obiekty `list`
cities = []  # Nazwy miejscowości
X = []  # Szerokość geograficzna
Y = []  # Długość geograficzna

# Wczytujemy dane z pliku
with open("miasta.csv", "r", encoding='utf-8') as dane:
    cities_all = dane.readlines()
```

Warto przyjrzeć się dokładniej konstrukcji służącej do odczytywania danych z pliku:

```
with open(_nazwa_pliku_, _atrybut_, _kodowanie_) as dane:
    # Tu blok kodu, który będzie wykonany
    # Po zakończeniu Python sam dba o zamknięcie pliku
```

Możliwe atrybuty do odczytu pliku to:

Wartość	Znaczenie atrybutu
`'r'`	Otwarcie tylko do odczytu (domyślne działanie)
`'b'`	Tryb binarny
`'t'`	Tryb tekstowy (domyślne działanie)

Jak widać, w naszym przypadku używamy atrybutu `r`, choć nie musielibyśmy tego robić (tak jak nie używamy atrybutu `t`). Przy czytaniu danych z plików tekstowych mamy do dyspozycji trzy metody:

- `read()` — czyta cały plik do jednego dużego obiektu tekstowego; możemy użyć `read(size)`, wówczas wczytane zostanie tylko `size` bajtów;
- `readline()` — wczytanie jednej linii z pliku do jednego obiektu tekstowego;
- `readlines()` — wczytanie wszystkich linii z pliku tekstowego do obiektu listy, gdzie każda linia to kolejny element.

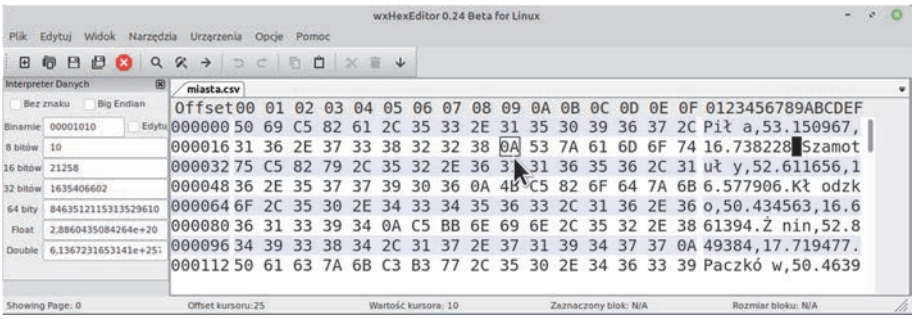
Uwaga!

Ponieważ domyślnie Windows używa strony kodowej CP-1250, wymusimy na nim odczyt pliku stworzonego z użyciem kodowania UTF-8, domyślnego w systemach Linux i macOS. Użyjemy w tym celu parametru `encoding='utf-8'`. Dzięki temu nasz program będzie działać poprawnie w każdym systemie operacyjnym.

Obiekt `cities_all` zawiera wszystkie linie wczytane z pliku w postaci listy; każdy jej element to jeden wiersz danych.

```
['Piła,53.150967,16.738228\n', 'Szamotuły,52.611656,16.577906\n',
'Kłodzko,50.434563,16.661394\n', 'Żnin,52.849384,17.719477\n',
'Paczków,50.463993,17.006659\n', 'Kazimierz Dolny,51.318043,21.954248\n',
'Bolesławiec,51.265854,15.565740\n', 'Sztutowo,54.326084,19.179071\n',
'Kudowa-Zdrój,50.442715,16.242661\n', 'Olkusz,50.281315,19.565687\n',
'Lubartów,51.459869,22.609461\n', 'Wieliczka,49.987061,20.064796\n',
'Police,53.552040,14.572061\n', 'Przemyśl,49.783863,22.767792\n',
'Myszyniec,53.380493,21.349531\n', 'Kielce,50.866077,20.628569\n',
'Frombork,54.357479,19.680073\n', 'Zakopane,49.299030,19.949047\n',
'Sopot,54.444092,18.570328\n', 'Krosno,49.690289,21.754990\n',
'Pruszków,52.170471,20.811886\n', 'Łeba,54.753063,17.557011\n', 'Kędzierzyn-
Koźle,50.349880,18.226185\n', 'Grodziec,52.038197,18.059839\n']
```

Zwróć uwagę na ważny element — tę dziwną sekwencję znaków `\n` na końcu każdego elementu listy. Skąd się tu wzięła? Przecież w pliku jej nie było… a może jednak była? Tak, była, po prostu jej nie zauważyliśmy. Przypatrzmy się zatem bliżej temu plikowi, wyświetlonemu w specjalnym edytorze (tzw. heksadecymalnym):



Rysunek 6.2. Plik z danymi wyświetlony w oknie edytora heksadecymalnego

Na rysunku 6.2 możemy zobaczyć (wskazany strzałką kursora) znak o kodzie `0A`; to w tablicy ASCII znak opisany jako *LF — Line Feed*. Na stronie Wikipedii poświęconej temu kodowi możemy przeczytać, że w różnych systemach operacyjnych ten znak ma różne wartości (ja zwykle pracuję w systemie Linux, więc dlatego w edytorze widać właśnie `0A`).

W różnych systemach operacyjnych stosuje się określone znaki końca linii, oparte na kombinacji kodów CR (ang. *carriage return*, szesnastkowo `0x0D`) i LF (ang. *line feed*, szesnastkowo `0x0A`):

- `LF` — w systemach Unix, BeOS, AmigaOS, MorphOS, RISC OS, GNU/Linux, macOS, Multics;
- `CRLF` — w systemach DOS, OS/2, Microsoft Windows, Symbian, DEC RT-11;
- `CR` — w systemach Commodore, Apple II, Mac OS (do wersji 9), Microware OS-9.

Już wiesz, skąd się biorą te znaki. Czy będą nam przeszkadzały? Niezbyt, za chwilę zresztą się ich pozbędziemy. Każdy element pliku rozkładamy w pętli na 3-elementową listę, używając w tym celu metody `split()`, która pozwala na podzielenie ciągu znaków z użyciem zdefiniowanego separatora. Domyślnie separatorem jest znak spacji, o ile nie podamy jawnie innego.

```
# Czyścimy dane
for city in cities_all:
    datas = city.split(
        ","
    )  # Tu dzielimy tekst na elementy listy, znak `,` jest
separatorem
    cities.append(datas[0].strip())  # Dodajemy do listy nazwę
miejscowości
    X.append(float(datas[1]))  # i kolejne wartości współrzędnych
    Y.append(float(datas[2]))
```

Na końcu wyświetlimy dla pewności nasze dane. Użyjemy w dość nietypowy sposób funkcji `print()`. Zastosujemy parametr `sep` (czyli separator). Pozwoli on nam wyświetlić 3 listy, każdą w kolejnej linii, ale z użyciem jednego wywołania funkcji.

```
# Teraz zobaczymy nasze dane
print(cities, X, Y, sep="\n======\n")
# Analogiczny efekt uzyskamy stosując polecenia:
# print(cities)
# print("======")
# print(X)
# print("======")
# print(Y)
```

Możemy sprawdzić działanie tych poleceń, usuwając sprzed nich kolejno znak komentarza. Warto to zrobić, by mieć świadomość, ile oszczędzamy czasu, choćby na pisanie kolejnych „printów" — w przypadku ograniczonego czasu (np. na maturze) warto znać takie sztuczki.

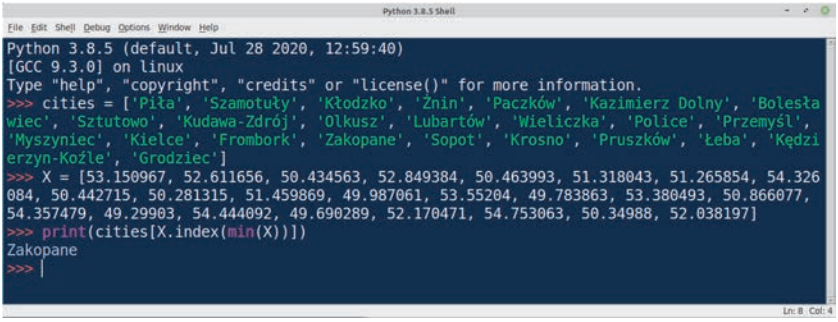
Oto efekt działania tego kodu:

```
Moduł cartopy wczytany.
Moduł matplotlib wczytany.
['Piła', 'Szamotuły', 'Kłodzko', 'Żnin', 'Paczków', 'Kazimierz Dolny',
'Bolesławiec', 'Sztutowo', 'Kudowa-Zdrój', 'Olkusz', 'Lubartów',
'Wieliczka', 'Police', 'Przemyśl', 'Myszyniec', 'Kielce', 'Frombork',
'Zakopane', 'Sopot', 'Krosno', 'Pruszków', 'Łeba', 'Kędzierzyn-Koźle',
'Grodziec']
======
[53.150967, 52.611656, 50.434563, 52.849384, 50.463993, 51.318043,
51.265854, 54.326084, 50.442715, 50.281315, 51.459869, 49.987061, 53.55204,
49.783863, 53.380493, 50.866077, 54.357479, 49.29903, 54.444092, 49.690289,
52.170471, 54.753063, 50.34988, 52.038197]
======
[16.738228, 16.577906, 16.661394, 17.719477, 17.006659, 21.954248, 15.56574,
19.179071, 16.242661, 19.565687, 22.609461, 20.064796, 14.572061, 22.767792,
21.349531, 20.628569, 19.680073, 19.949047, 18.570328, 21.75499, 20.811886,
17.557011, 18.226185, 18.059839]
```

Mamy zatem listy z kolejnymi elementami. Wyobraź sobie, że tego rodzaju dane zawiera zadanie maturalne, brzmiące:

Znajdź miasto położone najdalej na południe.

Co wówczas możesz zrobić? To bardzo proste — znajdujesz najmniejszy element w liście `X` (szerokość geograficzna) i odnajdujesz indeks tej wartości, a następnie sprawdzasz nazwę miejscowości, np.:



Rysunek 6.3. Wyszukiwanie miasta położonego najdalej na południe — realizowane w środowisku Python IDLE

```
print(cities[X.index(min(X))])
Zakopane
```

Oczywiście aby to zadziałało, każde z wymienionych miast musi być położone na innym równoleżniku, lecz zazwyczaj w pytaniach maturalnych dane są odpowiednio przygotowane, więc nie trzeba się tym martwić.

6.3. Przygotowanie mapy

Wróćmy teraz do naszego głównego zadania — rysowania. Mamy dane, więc zajmijmy się przygotowaniem bazowej mapy.

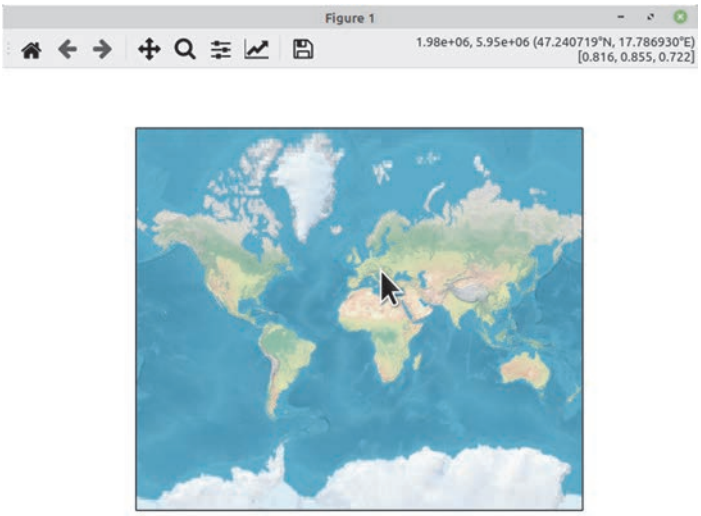
```
# program r6_02_mapa.py
# Podstawowa mapa

# Tworzymy okno
figure = plt.figure(figsize=(7, 5))
ax = figure.add_subplot(1, 1, 1, projection=crs.Mercator())  # Dodajemy
projekcję Merkatora

# Dodajemy właściwość do mapy - zdjęcie
ax.stock_img()

# Wyświetlamy okno
plt.show()
```

Wszystkie nasze działania będziemy wykonywać w odniesieniu do obiektu `ax`; np. metoda, której używamy, powoduje pojawienie się pięknego obrazka jako tła. I mamy naszą podstawową mapę!



Rysunek 6.4. Podstawowa mapa

My spróbujemy tutaj wyświetlić Europę, a więc podamy wycinek od –10° długości geograficznej E, do 35° długości geograficznej E oraz od 66° szerokości geograficznej N do 34° szerokości geograficznej N. Jeśli masz ochotę, możesz wybrać inne wartości; najlepiej zrób to kilka razy i obserwuj zmiany.

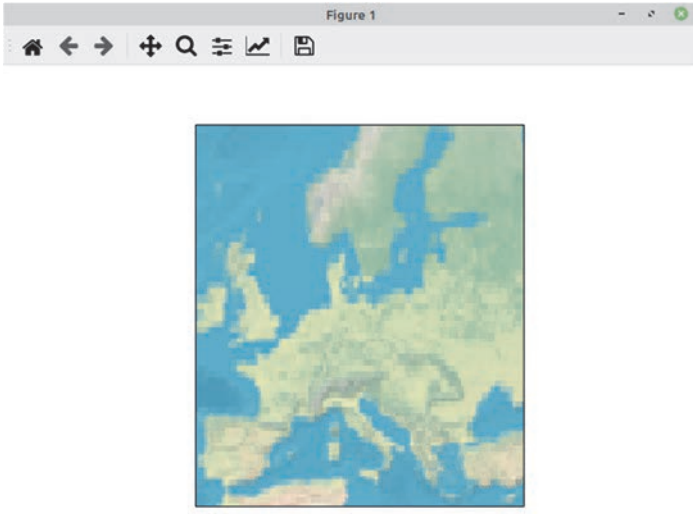
```
# program r06_03_mapa_eu.py
# Mapa Europy

# Dodajemy właściwość do mapy - zdjęcie
ax.stock_img()

# Wydzielamy tylko wycinek mapy - Europę
ax.set_extent([-10, 35, 66, 34], crs=crs.PlateCarree())

# Wyświetlamy okno
plt.show()
```

Nasza grafika będzie wyglądała następująco:



Rysunek 6.5. Mapa Europy

Teraz przychodzi moment oznaczenia miast. Użyjemy znanej już nam metody `scatter()`, a w miejsce wartości `X` damy listę długości geograficznych, dla `Y` damy wartości szerokości geograficznych. Kolor punktów ustawimy na czerwony, a rozmiar — na 4 piksele.

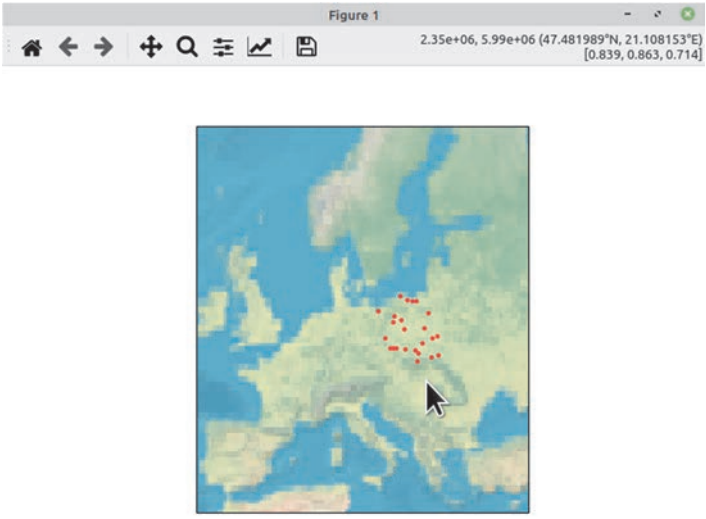
```
# program r06_03_mapa_punkty.py
# Mapa ostateczna

# Wydzielamy tylko wycinek mapy - Europę
ax.set_extent([-10, 35, 66, 34], crs=crs.PlateCarree())

# Oznaczamy punkty
plt.scatter(x=Y, y=X, color="red", s=4, alpha=1,
transform=crs.PlateCarree())

# Wyświetlamy okno
plt.show()
```

Efekt może nie jest porażający, jednak trzeba pamiętać, że uzyskaliśmy go niewielkim nakładem pracy.



Rysunek 6.6. Gotowa mapa

Mamy to!

Pełny kod

Teraz zobaczmy pełny kod naszego programu:

```
# program r06_03_mapa_punkty.py
# Mapa ostateczna

from sys import exit

try:
    import cartopy.crs as crs
    import cartopy.feature as cfeature

    print("Moduł cartopy wczytany.")
except:
    print("Zainstaluj: 'pip install cartopy' ")
    exit(0)

try:
    import matplotlib.pyplot as plt

    print("Moduł matplotlib wczytany.")
except:
    print("Zainstaluj: 'pip install matplotlib' ")
    exit(0)
```

```
# Wczytane dane będziemy zapisywać jako obiekty `list`
cities = []  # Nazwy miejscowości
X = []  # Szerokość geograficzna
Y = []  # Długość geograficzna

# Wczytujemy dane z pliku
with open("miasta.csv", "r", encoding='utf-8') as dane:
    cities_all = dane.readlines()

print(cities_all)

# Czyścimy dane
for city in cities_all:
    datas = city.split(",")
    cities.append(datas[0].strip())
    X.append(float(datas[1]))
    Y.append(float(datas[2]))

# Teraz zobaczymy nasze dane
print(cities, X, Y, sep="\n======\n")

# Tworzymy okno
figure = plt.figure(figsize=(7, 5))
ax = figure.add_subplot(
    1, 1, 1, projection=crs.Mercator()
)  # Dodajemy projekcję Merkatora

# Dodajemy właściwość do mapy - zdjęcie
ax.stock_img()

# Wydzielamy tylko wycinek mapy - Europę
ax.set_extent([-10, 35, 66, 34], crs=crs.PlateCarree())

# Oznaczamy punkty
plt.scatter(x=Y, y=X, color="red", s=4, alpha=1,
transform=crs.PlateCarree())

# Wyświetlamy okno
plt.show()
```

Pamiętaj!

Nigdy nie rezygnuj z marzenia tylko dlatego, że zrealizowanie go wymaga czasu. Czas i tak upłynie — Earl Nightingale

Twórz różne programy w Pythonie — i baw się świetnie!

Jeśli:

- **chcesz poznać język Python od strony praktycznej**
- **przymierzasz się do matury z informatyki**
- **marzysz o karierze programisty**

to doskonale trafiłeś!

Dzięki tej książce przekonasz się, jak wspaniałą przygodą jest programowanie i jak łatwo ją zacząć! Poznasz podstawy Pythona, dowiesz się, jak pisać i formatować kod, a także szybko nauczysz się uruchamiać swoje programy. Instrukcje sterujące, operatory, typy danych, funkcje, klasy i moduły nie będą miały przed Tobą tajemnic, a to jeszcze nie koniec! Przede wszystkim będziesz poznawać Pythona od strony praktycznej, tworząc projekty prawdziwych gier i symulacji oraz aplikacje do wizualizacji danych i anonimizowania metadanych plików graficznych.

Możesz użyć tej książki jako pomocy w przygotowaniu do matury i wsparcia w wyborze drogi zawodowej. Przekonaj się, że nauka może być najlepszą zabawą. Baw się dobrze i zdaj egzamin celująco — oczywiście z Pythonem!

- **środowisko IDLE**
- **podstawy Pythona w wersji 3.6 i wyższej**
- **konstrukcje języka**
- **projekty gier**
- **symulacje fizyczne**
- **prezentacja i wizualizacja danych**
- **praktyczne zastosowania Pythona**

Okiełznaj Pythona i naucz się programować!

Adam Jurkiewicz — programista i administrator systemów UNIX/Linux z ponad 30-letnim doświadczeniem. Zdobywca wyróżnienia Szerokiego Porozumienia na rzecz Umiejętności Cyfrowych w Polsce w latach 2017 i 2020, trener języka programowania Python, robotyki, mechatroniki, technologii komputerowych ze szczególnym uwzględnieniem otwartych zasobów edukacyjnych i oprogramowania *open source*. Współautor treści dotyczących języka Python w e-podręcznikach do kształcenia ogólnego dla klas ponadgimnazjalnych. Współautor książki *Koduj w Pythonie. Tworzymy grę przygodową* i publikacji *Enigma. Poznaj zagadkę Enigmy, tworząc grę przygodową w Pythonie*. Miłośnik szant, stateczny mąż, ojciec i dziadek, a także zwariowany nauczyciel młodzieży, jeśli tylko ma okazję.

Helion

helion.pl

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

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-289-0602-0

9 788328 906020

Cena: 69,00 zł