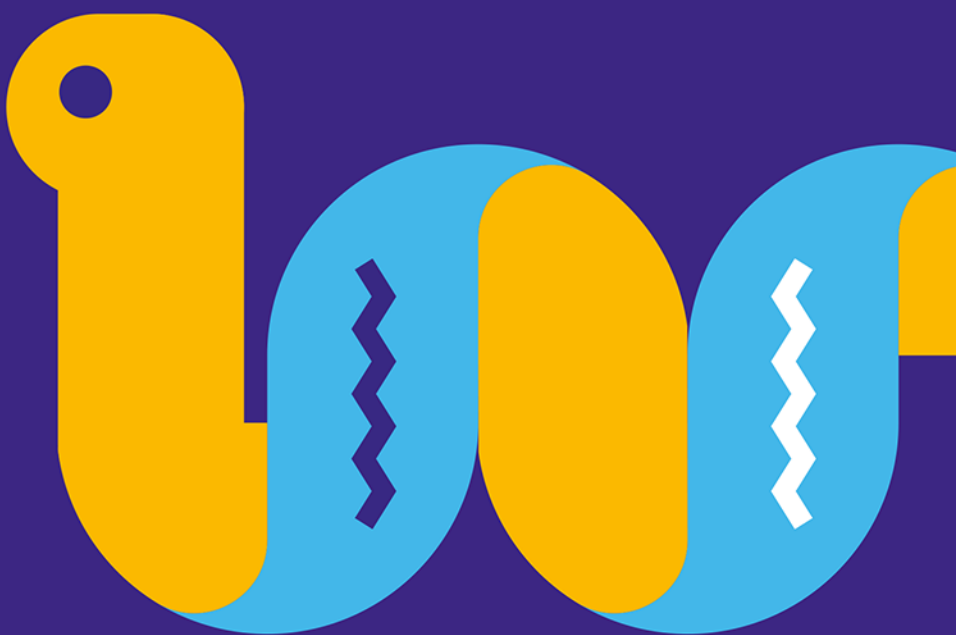
ZOFIA MATUSIEWICZ

ZACZNIJ OD PYTHONA

PROGRAMOWANIE DLA MŁODZIEŻY W PRAKTYCE



Helion

Redaktor prowadzący: Szymon Szwajger

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. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-283-9027-0

Printed in Poland.

Spis treści

Wstęp

Drogi Czytelniku, otwierasz niewielką książeczkę dotyczącą programowania. Zastanawiasz się pewnie, czy jest dla Ciebie, czy sobie z nią poradzisz, czy Cię zainteresuje. Jeżeli wziąłeś ją do ręki, to znaczy, że chcesz się uczyć programowania. To bardzo dobrze!

W dzisiejszych czasach spora część życia toczy się przed komputerem, niemal zawsze mamy go pod ręką. Stanowi już nie tylko narzędzie do pracy, ale również służy do relaksu. Większość urządzeń codziennego użytku również bazuje na komputerach — nierzadko programowalnych.

Umiejętność programowania daje nam szansę, by zrozumieć otaczający nas świat i nad nim zapanować. Im wcześniej zaczniesz naukę, tym lepiej dla Ciebie. Mam nadzieję, że pierwsze kroki w tym kierunku postawisz jeszcze w szkole 😉.

Kolejne etapy nauki programowania otwierają zawsze przed nami nowe możliwości. Jeżeli już znasz podstawowe polecenia języka Python, to ta książka jest właśnie dla Ciebie (jeśli nie, sięgnij najpierw po *Zacznij od Pythona. Pierwsze kroki w programowaniu*). To kolejny krok, który otworzy przed Tobą nowe możliwości i pokaże Ci, jak wiele rzeczy możesz już zrobić, ale pozostawi też niedosyt. Pokaże Ci, jak wiele jeszcze można, a właściwie trzeba się nauczyć.

Moim marzeniem jest przeprowadzić Cię przez ten nowy etap i rozbudzić w Tobie pragnienie, by przejść kolejny, kolejny i kolejny.

Pamiętaj jednak, że nie jest to typowy podręcznik. Nie chcę, byś podczas czytania tej książki czuł się jak w muzeum czy galerii sztuki — jak widz. Poczuj się jak uczestnik warsztatów, na których wszystkiego możesz dotknąć i wszystkiego spróbować. Nawet jeśli na początku nie wszystko idzie po Twojej myśli, działaj dalej.

Kilka uwag o książce

Celem tej książki jest poruszenie ważnego tematu, jakim są dane. One są tu głównym bohaterem. Zaczniemy od tego, czy można jeden typ danych zmieniać w inny, a jeżeli tak, to w jaki sposób. Powiemy sobie, jak ładnie wyświetlać dane, jak je wczytywać, zapisywać w pliku oraz jak tworzyć kolekcje danych. Jednak dane to nie tylko liczby, dlatego powiemy też, jak budować obiekty.

Jak zbudowana jest ta książka? Każdy rozdział zaczyna się od objaśnienia tematu, dalej wraz z instrukcjami do Pythona pojawiają się przykłady, które mają na celu rozjaśnić Ci zagadnienie. Każdy rozdział kończy się zestawem ćwiczeń — czasem bardzo prostych, by można było przećwiczyć nowe treści, a czasem podchwytliwych, byś mógł trochę podumać nad określonym zagadnieniem.

Programy piszemy oczywiście w Pythonie. W książce używamy środowiska Anaconda, które można pobrać ze strony *https://www.anaconda.com*. Programy zapisujemy w Jupyter Notebook. Zachęcam Cię do zainstalowania Anacondy, w której programy w języku Python możesz pisać także w środowiskach programistycznych takich jak PyCharm czy Spyder. Mam nadzieję, że również spróbujesz to robić.

Ponieważ książka jest skierowana do najmłodszego grona informatyków, czyli do dzieci i młodzieży, staram się — na ile to możliwe — unikać języka formalnego i symboliki matematycznej. Niektóre treści często się powtarzają, ponieważ zdaję sobie sprawę, że nie wszystko można od razu zapamiętać. A skupiając się na nowym temacie, nie chcemy tracić wątku,

więc nie cofamy się, by uzupełnić wiedzę. Jeśli któryś z rozdziałów będzie dla Ciebie nie całkiem zrozumiały, przeczytaj go i przejdź do następnego. Niektóre treści stają się jasne dopiero w zestawieniu z innymi zagadnieniami.

Rozdział 1.

Rzutowanie — konwersja typu

Każdy, kto pisze nawet najprostsze programy, wie, że Python jest językiem bardzo przyjaznym i dającym wiele możliwości. Posiada także wiele uproszczeń. Jednym z nich jest to, że nie musimy deklarować typu. Python sam „przydziela" typ zmiennej na podstawie wpisanej do niej wartości. W tym rozdziale pokażemy jednak coś więcej — omówimy sposoby zamiany typów. Ponadto zwrócimy uwagę na rzutowanie, które jest wynikiem przetwarzania danych. Pokażemy, jaka konwersja jest dostępna w Pythonie, a jaka powoduje błędy.

Znamy już kilka funkcji, które pozwalają zmieniać typ podanej wartości:

- `int(`*`wartość`*`)` — na całkowity,
- `float(`*`wartość`*`)` — na zmiennoprzecinkowy,
- `bool(`*`wartość`*`)` — na logiczny,
- `str(`*`wartość`*`)` — na tekstowy.

Omówimy teraz różne wersje konwersji.

Typ całkowity

Zacznijmy od zamiany różnych wartości na typ całkowity. Zaczniemy od przykładów ilustrujących, które typy danych zamieniają się na liczbę całkowitą.

Przykład 1. Zobaczmy zamianę liczby zmiennoprzecinkowej („z ułamkiem") na liczbę całkowitą:

```
liczba = 14.6
liczba = int(liczba)
print(liczba, type(liczba))
```

Konwersja spowodowała obcięcie części ułamkowej. Nie jest to jednak zaokrąglenie podanej liczby:

```
14 <class 'int'>
```

Przykład 2. Funkcja `round` służy do zamiany liczby zmiennoprzecinkowej na całkowitą poprzez jej zaokrąglenie:

```
liczba = 14.6
zaokrąglenie = round(liczba)
print(zaokrąglenie, type(zaokrąglenie))
```

Wynikiem jest:

```
15 <class 'int'>
```

Przykład 3. Przypomnijmy jeszcze, że `3.14` jest liczbą zmiennoprzecinkową, zaś `3,14` jest krotką/tuplą (o krotkach więcej będzie również w dalszej części książki. Tutaj podkreślmy tylko, że `3,14` to para dwóch liczb: `3` oraz `14`).

```
liczba = 3,14
nowa = int(liczba)
print(nowa, typ(nowa))
```

Powyższy kod zwróci błąd:

```
TypeError Traceback (most recent call last)
<ipython-input-26-d74237b78d46> in <module>
      1 liczba = 3,14
----> 2 nowa = int(liczba)
      3 print(nowa, typ(nowa))

TypeError: int() argument must be a string, a bytes-like object or
↪a number, not 'tuple'
```

Przykład 4. Pamiętajmy, że przemnażanie liczby przez 10, 100, 1000 (potęgi liczby 10), nawet jeżeli zwróci wartość całkowitą, nie zmienia jej typu:

```
liczba = 14.6
nowa = liczba*100
print(nowa, type(nowa))
```

i zwraca:

```
1460.0 <class 'float'>
```

Przykład 5. Zamiana wartości logicznej na liczbę całkowitą:

```
prawda = True
liczba = int(prawda)
print(liczba, type(liczba))
```

gdzie wynikiem jest:

```
1 <class 'int'>
```

oraz drugiej wartości logicznej:

```
fałsz = False
liczba = int(fałsz)
print(liczba, type(liczba))
```

co daje:

```
0 <class 'int'>
```

Konwersja przebiegła pomyślnie i zwraca 1, gdy wartość jest `True`, lub 0, gdy wartość jest `False`.

Przykład 6. Zamiana tekstu wyglądającego jak liczba całkowita na liczbę całkowitą:

```
tekst = '3'
liczba = int(tekst)
print(liczba, type(liczba))
```

Konwersja przebiegła pomyślnie:

```
3 <class 'int'>
```

Przykład 7. Zamiana tekstu wyglądającego jak liczba wymierna (z częścią ułamkową) na liczbę całkowitą:

```
tekst = '5.1'
liczba = int(tekst)
print(liczba, type(liczba))
```

Próba konwersji zwróciła błąd:

```
ValueError Traceback (most recent call last)
<ipython-input-3-d6d4685d53b0> in <module>
      1 tekst = '5.1'
----> 2 liczba = int(tekst)
      3 print(liczba, type(liczba))

ValueError: invalid literal for int() with base 10: '5.1'
```

Jeżeli chcemy zamienić ten tekst na liczbę całkowitą, możemy to zrobić w następujący sposób:

```
tekst = '5.1'
liczba = float(tekst)
liczba = int(liczba)
print(liczba, type(liczba))
```

Da to pomyślny rezultat:

```
5 <class 'int'>
```

Powyższy program możemy uprościć, **składając funkcje** (wtedy argumentem jednej funkcji jest wynik innej):

```
tekst = '5.1'
liczba = int(float(tekst))
print(liczba, type(liczba))
```

Przykład 8. Podamy teraz przykład zamiany tekstu na liczbę całkowitą, która nie jest rzutowaniem, ale podmianą wartości (a wraz z nią typu):

```
liczba = input("Podaj liczbę rzymską od I do X")
print(liczba, type(liczba))
if liczba == "I":
    liczba = 1
elif liczba == "II":
    liczba = 2
elif liczba == "III":
    liczba = 3
elif liczba == "IV":
    liczba = 4
elif liczba == "V":
    liczba = 5
elif liczba == "VI":
    liczba = 6
elif liczba == "VII":
    liczba = 7
elif liczba == "VIII":
    liczba = 8
elif liczba == "IX":
    liczba = 9
elif liczba == "X":
    liczba = 10
else:
    liczba = 0
print(liczba, type(liczba))
```

Dla przykładu może to zwrócić:

```
Podaj liczbę rzymską od I do XIII
III <class 'str'>
3 <class 'int'>
```

Przykład 9. W wielu językach programowania (np. C++) jest możliwość konwersji typu znakowego na liczbę całkowitą. Dla wspomnianego C++ kod:

```
int x;
x = 21 +'A';
```

zwróci `x = 86` (gdyż w kodach ASCII `'A'` jest równe 65). Natomiast odpowiedni kod w Pythonie:

```
x = 21 +'A'
```

zwróci następujący błąd:

```
TypeError Traceback (most recent call last)
↪<ipython-input-18-d6d137685555> in <module>
----> 1 x = 21 +'A'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Dla języka Python konwersji z kodu ASCII trzeba dokonać za pomocą funkcji `ord`:

```
x = 21 +ord('A')
print(x)
```

Otrzymamy wtedy (tak samo jak dla C++) `x = 86`.

Typ zmiennoprzecinkowy

Kontynuując temat typów liczbowych, przyjrzyjmy się, jak działa funkcja `float` i jakie działania spowodują zrzutowanie na typ `float`.

Przykład 10. Zacznijmy od zamiany liczby całkowitej na zmiennoprzecinkową:

```
liczba = 7
nowa = float(liczba)
print(nowa, type(nowa))
```

Zwróćmy uwagę, że liczba zmiennoprzecinkowa, nawet nieposiadająca części ułamkowej, pisana jest z kropką. Tutaj mamy:

```
7.0 <class 'float'>
```

Przykład 11. Zobaczmy teraz, że różne działania zwracają wartości typu `float`:

```
liczba_1 = 12
liczba_2 = 4
liczba_3 = liczba_1/liczba_2
print(liczba_1,type(liczba_1), liczba_2,type(liczba_2),
↪liczba_3,type(liczba_3))
```

Oznacza to, że dzielenie na liczbach całkowitych zwraca `float`:

```
12 <class 'int'> 4 <class 'int'> 3.0 <class 'float'>
```

Podobnie jest w przypadku pierwiastkowania czy potęgowania z wykładnikiem niecałkowitym:

```
import math
liczba_1 = 16
liczba_2 = math.sqrt(liczba_1) #pierwiastek
liczba_3 = pow(16, 0.25)
print(liczba_1,type(liczba_1), liczba_2,type(liczba_2),
↪liczba_3,type(liczba_3))
```

Tutaj otrzymujemy:

```
16 <class 'int'> 4.0 <class 'float'> 2.0 <class 'float'>
```

Przykład 12. Zobaczmy teraz, że konwersja przebiega pomyślnie bez względu na to, czy liczba w tekście ma część ułamkową, czy nie:

```
tekst_1 = '34'
liczba_1 = float(tekst_1)
tekst_2 = '12.5'
liczba_2 = float(tekst_2)
print(liczba_1, type(liczba_1), liczba_2,type(liczba_2))
```

Tutaj otrzymujemy:

```
34.0 <class 'float'> 12.5 <class 'float'>
```

Przykład 13. Zamienimy teraz wartości typu logicznego na `float`:

```
liczba_1 = float(True)
liczba_2 = float(False)
print(liczba_1,type(liczba_1), liczba_2,type(liczba_2))
```

Zgodnie z intuicją otrzymujemy wartości `1.0` oraz `0.0`:

```
1.0 <class 'float'> 0.0 <class 'float'>
```

Przykład 14. Zauważmy, że skoro możemy wykonywać działania na wartościach logicznych, to ich rezultaty też mogą zmienić typy:

```
liczba_1 = 2*True +True/(2*True)
print(liczba_1,type(liczba_1))
```

A daje to wynik:

```
2.5 <class 'float'>
```

Elegancko możemy to także zapisać następująco:

```
liczba_1 = float(2*True +True/(2*True))
print(liczba_1,type(liczba_1))
```

Przykład 15. Dodajmy również, że nie nastąpi automatyczne rzutowanie typu `float` na tekstowy:

```
liczba = 7.0
liczba = liczba+'2'
```

Da to błąd:

```
TypeError Traceback (most recent call last)
↪<ipython-input-44-bf56a6661346> in <module>
      1 liczba = 7.0
----> 2 liczba = liczba+'2'
```

Nie nastąpi również automatyczne rzutowanie z typu tekstowego na `float`:

```
znak = '4'
znak = znak + 5
```

bo zwróci błąd:

```
TypeError Traceback (most recent call last)
↪<ipython-input-46-b24ce326fd2a> in <module>
      1 znak = '4'
----> 2 znak = znak + 5

TypeError: can only concatenate str (not "int") to str
```

Typ logiczny

Zobaczmy teraz, czy i jak różne wartości liczbowe i tekstowe zamieniają się na wartości logiczne: `True` i `False`.

Przykład 16. Zamieńmy dowolną liczbę całkowitą na typ logiczny:

```
liczba = 1
wartosc_logiczna = bool(liczba)
```

```
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = 0
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = 34
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = -23
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
```

Zaobserwujmy, że wartości niezerowe zwracają `True`, zaś zero konwertuje się na `False`.

Zobaczmy teraz, że podobna sytuacja występuje w przypadku liczb zmiennoprzecinkowych.

Przykład 17. Dla liczb wymiernych również następuje konwersja na typ logiczny:

```
liczba = 0.0
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = 0.02
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = -0.007
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = 1.0
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = 0.99
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = 3.4
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
liczba = -2.3
wartosc_logiczna = bool(liczba)
print(wartosc_logiczna, type(wartosc_logiczna))
```

i także zwraca `False` tylko dla `0.0`:

```
False <class 'bool'>
True <class 'bool'>
True <class 'bool'>
True <class 'bool'>
True <class 'bool'>
True <class 'bool'>
True <class 'bool'>
```

Typ tekstowy

Zamiana dowolnego typu danych na tekst wydaje się nie tylko prostsza i bardziej oczywista, ale również — jak się później okaże — bardzo ważna.

Przykład 18. Zamianę dowolnego typu na tekst możemy zrobić dzięki metodzie `str`. Zobaczmy program z kilkoma przykładami:

```
liczba = 3
tekst = str(liczba)
print(liczba, type(liczba), tekst, type(tekst))
liczba = 4.3
tekst = str(liczba)
print(liczba, type(liczba), tekst, type(tekst))
prawda = True
tekst = str(liczba)
print(liczba, type(liczba), prawda, type(prawda))
```

Otrzymujemy wynik:

```
3 <class 'int'> 3 <class 'str'>
4.3 <class 'float'> 4.3 <class 'str'>
4.3 <class 'float'> True <class 'bool'>
```

Przykład 19. Przypomnijmy również, że wczytanie zwraca typ tekstowy:

```
bok_prostokąta_1 = input("Podaj długość boku prostokąta a = ")
bok_prostokąta_2 = input("Podaj długość boku prostokąta b = ")
obwód = 2*bok_prostokąta_1 + 2*bok_prostokąta_2
print(obwód, type(obwód))
```

co daje przykładowo wynik:

```
Podaj długość boku prostokąta a = 4
Podaj długość boku prostokąta b = 5
4455 <class 'str'>
```

Poprawny kod będzie zatem następujący:

```
bok_prostokąta_1 = float(input("Podaj długość boku prostokąta a = "))
bok_prostokąta_2 = float(input("Podaj długość boku prostokąta b = "))
obwód = 2*bok_prostokąta_1 + 2*bok_prostokąta_2
print(obwód, type(obwód))
```

Otrzymujemy wówczas poprawny wynik:

```
Podaj długość boku prostokąta a = 4
Podaj długość boku prostokąta b = 5
18.0 <class 'float'>
```

Przykład 20. Inne struktury danych też mają napisaną funkcję `str`:

```
tablica = ["ala", "ma", "kota"]
print(tablica)
krotka = (1, 2, 3)
print(krotka)
słownik ={"be":"być", "can":"móc", "cat":"kot"}
print(słownik)
zbiór = {"a", "b", "c"}
print(zbiór)
```

Otrzymujemy:

```
['ala', 'ma', 'kota']
(1, 2, 3)
{'be': 'być', 'can': 'móc', 'cat': 'kot'}
{'b', 'c', 'a'}
```

Zadania do samodzielnego rozwiązania

Zadanie 1

Podaj, co będzie wynikiem kodu i jaki typ zwróci:

a) `int(3.67)`,

b) `round(34.56)`,

c) `bool(3)`,

d) `bool(1)`,

e) `float(3.45)`,

f) `float(7,89)`,

g) `str('45')+'77'`,

h) `str('4')+5`,

i) `int('45.6')`,

j) `int('67')`,

k) `int('siedem')`,

l) `float('pi')`,

m) `int(4.5)+float(7)`,

n) `int(True)+2*int(True)`,

o) `bool(float(True)-1)`,

p) `65 + 'b'`,

q) `'7'+ 100`,

r) `'45'+'12'`.

Zadanie 2

Napisz program, który wczyta cyfrę w postaci tekstu (np. `dwa`) i zamieni ją na liczbę całkowitą.

Zadanie 3

Napisz program, który będzie wczytywał dowolne liczby zmiennoprzecinkowe i zwracał:

a) część całkowitą,

b) część ułamkową.

Zadanie 4

Sprawdź, jaki typ będą miały wyniki działań (wskaż miejsca, w których pojawi się błąd):

a) `9//4`,

b) `5/6`,

c) `round(64.2)/8`,

d) `int('3')+7`,

e) `int('25')+int('7.6')`,

f) `int(2.6)+int(3.4)`,

g) `10**5/10`,

h) `(10*int(True) – 7)/3`.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

DANE TO POTĘGA!

Skoro sięgasz po tę książkę, pewnie chcesz się uczyć programowania. To świetnie! Ta umiejętność z pewnością Ci się przyda — choćby do tego, by już dziś znakomicie się bawić „w towarzystwie" komputera, ale też jako inwestycja w przyszłość, podjęta z myślą o studiach i pracy.

Jednym z najpopularniejszych, a równocześnie dość prostych do nauki języków programowania jest Python. Możliwe, że nie jest Ci obcy. Jeśli jednak stykasz się z nim po raz pierwszy, podstawowe komendy tego języka możesz opanować między innymi dzięki poprzedniej książce Zofii Matusiewicz — *Zacznij od Pythona. Pierwsze kroki w programowaniu*.

Z KOLEJNEJ POZYCJI PRZYGOTOWANEJ PRZEZ AUTORKĘ DOWIESZ SIĘ SPORO NA TEMAT DANYCH, MIĘDZY INNYMI:

- **jak określony typ danych zmienić w inny**
- **jak ładnie wyświetlać dane**
- **jak je wczytywać i zapisywać w pliku**
- **jak tworzyć kolekcje danych**

UWAGA! PONIEWAŻ DANE TO NIE TYLKO LICZBY, PRZYJRZYMY SIĘ TAKŻE TEMU, JAK NA PRZYKŁAD BUDOWAĆ OBIEKTY.

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 978-83-283-9027-0

9 788328 390270

Cena: 37,00 zł