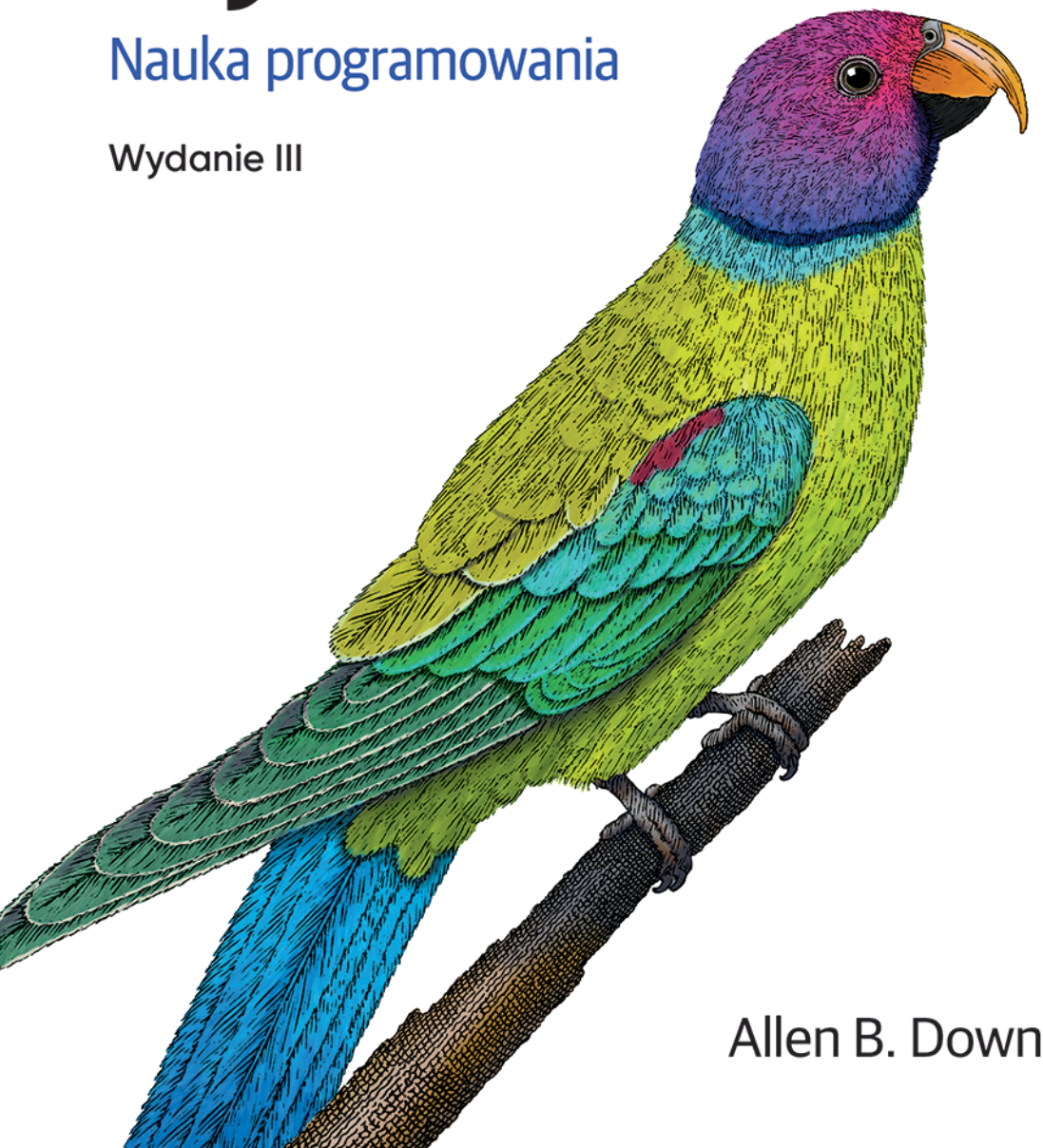
O'REILLY®

Helion

Myśl w języku Python!

Nauka programowania

Wydanie III



Allen B. Downey

Tytuł oryginału: Think Python: How to Think Like a Computer Scientist, 3rd Edition

Tłumaczenie: Piotr Pilch

Cytaty ze skeczów Monty Pythona w tłumaczeniu Tomasza Beksińskiego.

ISBN: 978-83-289-1902-0

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/myjep3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/myjep3.zip

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)

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Programowanie jako sposób rozumowania

Pierwszym celem tej książki jest nauczenie Cię sztuki programowania z użyciem języka Python. Uczenie się programowania oznacza jednak zaznajomienie się z nowym sposobem rozumowania, dlatego drugim celem książki jest ułatwienie Ci myślenia jak informatyk. Ten sposób rozumowania łączy w sobie niektóre najlepsze elementy matematyki, inżynierii i nauk przyrodniczych. Tak jak matematycy, informatycy używają języków formalnych do opisu idei (dokładniej rzecz biorąc, obliczeń). Tak jak inżynierowie, informatycy projektują różne rzeczy, łącząc komponenty w systemy i oceniając alternatywne warianty w celu znalezienia kompromisu. Podobnie do naukowców informatycy obserwują zachowanie złożonych systemów, stawiają hipotezy i sprawdzają przewidywania.

Zaczniemy od najbardziej podstawowych elementów programowania i będziemy stopniowo poszerzać zakres działań. W tym rozdziale dowiesz się, jak język Python reprezentuje liczby, litery i słowa. Nauczysz się również wykonywać operacje arytmetyczne.

Zaczniesz się też zaznajamiać ze słownictwem związanym z programowaniem, w tym z takimi terminami jak operator, wyrażenie, wartość i typ. Słownictwo to jest istotne — będzie Ci ono niezbędne do zrozumienia reszty zawartości książki, komunikowania się z innymi programistami oraz stosowania asystentów wirtualnych i pojęcia zasad ich działania.

Operatory arytmetyczne

Operator arytmetyczny to symbol reprezentujący obliczenie arytmetyczne. Przykładowo znak plusa (+) powoduje wykonanie operacji dodawania:

```
30 + 12
42
```

Znak minusa (–) to operator umożliwiający wykonanie operacji odejmowania:

```
43 - 1
42
```

Znak gwiazdki (*) powoduje wykonanie operacji mnożenia:

```
6 * 7
42
```

Ukośnik (/) służy do wykonania operacji dzielenia:

```
84 / 2
42.0
```

Zauważ, że wynik dzielenia to `42.0`, a nie `42`. Wynika to z faktu, że w języku Python występują następujące dwa typy liczb:

- **liczby całkowitoliczbowe**, które reprezentują liczby całkowite;
- **liczby zmiennoprzecinkowe**, które reprezentują liczby ze znakiem dziesiętnym.

Jeśli wykonujesz operacje dodawania, odejmowania lub mnożenia dwóch liczb całkowitych, wynik będzie liczbą całkowitą. Jeżeli jednak podzielisz dwie liczby całkowite, wynik będzie liczbą zmiennoprzecinkową. Język Python zapewnia kolejny operator (//), który wykonuje **dzielenie całkowitoliczbowe**. Wynikiem takiej operacji zawsze jest liczba całkowita:

```
84 // 2
42
```

Dzielenie całkowitoliczbowe jest również określane mianem „dzielenia podłogowego” (ang. *floor division*), ponieważ zawsze powoduje zaokrąglanie w dół (czyli w kierunku „podłogi”):

```
85 // 2
42
```

I na koniec, operator ** wykonuje operację potęgowania, czyli podnosi liczbę do potęgi:

```
7 ** 2
49
```

W niektórych innych językach znak ^ stosuje się do wykonywania potęgowania, ale w języku Python jest to operator bitowy zwany `XOR`. Jeśli nie jesteś zaznajomiony z operatorami bitowymi, wynik może być zaskakujący:

```
7 ^ 2
5
```

Nie będę omawiać w tej książce operatorów bitowych, ale możesz o nich przeczytać na stronie dostępnej pod adresem *http://wiki.python.org/moin/BitwiseOperators*.

Wyrażenia

Zbiór operatorów i liczb jest określany mianem **wyrażenia**. Wyrażenie może zawierać dowolną liczbę operatorów i liczb. Oto na przykład wyrażenie zawierające dwa operatory:

```
6 + 6 ** 2
42
```

Zauważ, że potęgowanie następuje przed dodawaniem. W języku Python przestrzega się kolejności operacji, o której miałeś okazję uczyć się na lekcjach matematyki: potęgowanie ma miejsce przed operacjami mnożenia i dzielenia, które są wykonywane przed dodawaniem i odejmowaniem.

W następującym przykładzie mnożenie następuje przed dodawaniem:

```
12 + 5 * 6
42
```

Jeśli chcesz, aby najpierw wykonano dodawanie, możesz użyć nawiasów okrągłych:

```
(12 + 5) * 6
102
```

Każde wyrażenie posiada **wartość**. Na przykład wyrażenie `6 * 7` zapewnia wartość `42`.

Funkcje arytmetyczne

Oprócz operatorów arytmetycznych język Python zapewnia kilka **funkcji** przetwarzających liczby. Na przykład funkcja `round` zaokrągla liczbę zmiennoprzecinkową do najbliższej liczby całkowitej:

```
round(42.4)
42
round(42.6)
43
```

Funkcja `abs` oblicza wartość bezwzględną liczby. W przypadku liczby dodatniej wartość bezwzględna to ta sama liczba:

```
abs(42)
42
```

Dla liczby ujemnej wartość bezwzględna jest dodatnia:

```
abs(-42)
42
```

Gdy funkcji używa się w ten sposób, ma się do czynienia z **wywoływaniem** funkcji. Wyrażenie, które wywołuje funkcję, to **wywołanie funkcji**.

Jeśli wywołujesz funkcję, nawiasy okrągłe są niezbędne. Jeżeli je pominiesz, zostanie wyświetlony następujący komunikat o błędzie:

```
abs 42
    Cell In[18], line 1
        abs 42
            ^
SyntaxError: invalid syntax
```

Możesz zignorować pierwszy wiersz tego komunikatu. Nie zawiera on żadnych informacji, które muszą być w tym momencie dla Ciebie zrozumiałe. Drugi wiersz to kod zawierający błąd z umieszczonym pod nim znakiem ^ w celu wskazania miejsca, w którym błąd został wykryty.

Ostatni wiersz wskazuje, że jest to **błąd składni** (ang. *syntax error*), co oznacza, że coś jest nie tak ze strukturą wyrażenia. W tym przykładzie problem polega na tym, że wywołanie funkcji wymaga nawiasów okrągłych.

Sprawdźmy, co się stanie, jeśli pominiesz nawiasy okrągłe *i* wartość:

```
abs
<function abs(x, /)>
```

Sama w sobie nazwa funkcji jest dozwolonym wyrażeniem z wartością. W momencie wyświetlenia jej wartość wskazuje, że `abs` to funkcja zawierająca dodatkowe informacje, które wyjaśnię później.

Łańcuchy

Oprócz liczb język Python może również reprezentować ciągi liter nazywane **łańcuchami**, ponieważ litery są ze sobą złączone jak koraliki naszyjnika. Aby utworzyć łańcuch znaków, można umieścić ciąg liter wewnątrz cudzysłowów prostych:

```
'Witaj, '
'Witaj, '
```

Dozwolone jest także zastosowanie cudzysłowów:

```
"świecie"
'świecie'
```

Cudzysłowy ułatwiają zapisanie łańcucha zawierającego apostrof, który jest tym samym symbolem co cudzysłów prosty:

```
"to mały świat"
"to mały świat"
```

Łańcuchy znaków mogą też zawierać spacje, znaki interpunkcyjne i cyfry:

```
'Cóż, '
'Cóż, '
```

Operator + działa z łańcuchami. Łączy on dwa łańcuchy w jeden, co jest określane mianem **konkatenacji**:

```
'Cóż, ' + "to mały " + 'świat.'
"Cóż, to mały świat."
```

Operator * współdziała także z łańcuchami. Tworzy on wiele kopii łańcucha i złącza je:

```
'Spam, ' * 4
'Spam, Spam, Spam, Spam, '
```

Inne operatory arytmetyczne nie obsługują łańcuchów.

Python zapewnia funkcję `len`, która oblicza długość łańcucha znaków:

```
len('Spam')
4
```

Zauważ, że funkcja `len` ustala liczbę liter między cudzysłowami, ale nie uwzględnia samych znaków cudzysłowu.

Tworząc łańcuch, pamiętaj o użyciu prostych cudzysłowów. Odwrotny apostrof, znany również jako *backtick*, powoduje błąd składni:

```
        `Witaj, `
Cell In[49], line 1
    `Witaj, `
    ^
SyntaxError: invalid syntax
```

Niedozwolone są również cudzysłowy inteligentne, nazywane także cudzysłowami falistymi.

Wartości i typy

Do tej pory zaprezentowano następujące trzy rodzaje wartości:

- `2` to liczba całkowita,
- `42.0` to liczba zmiennoprzecinkowa,
- `'Witaj, '` to łańcuch znaków.

Rodzaj wartości określany jest mianem **typu**. Każda wartość ma typ (czasem mówi się, że „należy ona do" typu).

Język Python zapewnia funkcję `type` informującą o tym, jaki jest typ dowolnej wartości. Typ liczby całkowitej to `int`:

```
type(2)
int
```

Typ liczby zmiennoprzecinkowej to `float`:

```
type(42.0)
float
```

Typ łańcucha znaków to `str`:

```
type('Witaj, świecie!')
str
```

Typy `int`, `float` i `str` mogą być używane jako funkcje. Przykładowo typ `int` może przyjąć postać liczby zmiennoprzecinkowej i przekształcić ją w liczbę całkowitą (zawsze zaokrąglając w dół):

```
int(42.9)
42
```

Z kolei typ `float` może zamienić liczbę całkowitą na liczbę zmiennoprzecinkową:

```
float(42)
42.0
```

Teraz coś, co może być niejasne. Co otrzymasz, jeśli umieścisz sekwencję cyfr wewnątrz cudzysłowów prostych?

```
'126'
'126'
```

Wygląda to jak liczba, ale w rzeczywistości jest to łańcuch:

```
type('126')
str
```

Jeśli spróbujesz użyć łańcucha jako liczby, możesz ujrzeć błąd:

```
'126' / 3
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

W tym przykładzie został wygenerowany błąd `TypeError`, co oznacza, że wartości w wyrażeniu, nazywane **argumentami**, mają niewłaściwy typ. Komunikat o błędzie wskazuje, że operator `/` nie obsługuje typów tych wartości, którymi są odpowiednio `str` i `int`.

W przypadku łańcucha zawierającego cyfry możesz użyć funkcji `int`, aby zamienić go na liczbę całkowitą:

```
int('126') / 3
42.0
```

Jeśli dysponujesz łańcuchem znaków zawierającym cyfry i znak dziesiętny, możesz użyć funkcji `float` w celu przekształcenia go w liczbę zmiennoprzecinkową:

```
float('12.6')
12.6
```

Przy zapisywaniu dużej liczby całkowitej może się pojawić pokusa, aby użyć przecinków między grupami cyfr, jak w przypadku liczby `1,000,000`. Jest to poprawne wyrażenie w języku Python, ale wynik nie jest liczbą całkowitą:

```
1,000,000
(1, 0, 0)
```

W języku Python liczba `1,000,000` jest interpretowana jako ciąg liczb całkowitych oddzielonych przecinkiem. W dalszej części rozdziału dowiesz się więcej o tego rodzaju ciągu.

Aby zwiększyć czytelność dużych liczb, możesz używać znaków podkreślenia:

```
1_000_000
1000000
```

Języki formalne i naturalne

Języki naturalne to języki, jakimi posługują się ludzie, takie jak angielski, hiszpański i francuski. Nie zostały stworzone przez ludzi, lecz rozwijały się w sposób naturalny.

Języki formalne to języki stworzone przez ludzi do konkretnych zastosowań. Na przykład notacja, jaką posługują się matematycy, jest językiem formalnym, który sprawdza się szczególnie dobrze przy opisywaniu relacji między liczbami i symbolami. Podobnie języki programowania są językami formalnymi zaprojektowanymi do definiowania obliczeń.

Choć języki formalne i naturalne mają wspólne elementy, występują między nimi następujące istotne różnice:

Wieloznaczność

Języki naturalne są pełne wieloznaczności, z którą ludzie radzą sobie, posługując się wskazówkami kontekstowymi oraz innymi informacjami. Języki formalne są tak projektowane, aby były prawie lub całkowicie jednoznaczne. Oznacza to, że każdy program, niezależnie od kontekstu, ma dokładnie jedno znaczenie.

Nadmiarowość

Aby zrekompensować wieloznaczność i zmniejszyć liczbę nieporozumień, w językach naturalnych występuje mnóstwo nadmiarowości. W rezultacie języki te często cechują się rozwlekłością. Języki formalne są mniej nadmiarowe i bardziej zwięzłe.

Dosłowność

Języki naturalne są pełne idiomów i metafor. W językach formalnych znaczenie instrukcji jest w pełni zgodne z jej treścią.

Ponieważ wszyscy dorastamy, posługując się językami naturalnymi, czasami trudno nam przyzwyczaić się do języków formalnych. Języki formalne są bardziej treściwe niż języki naturalne, dlatego w przypadku pierwszych z wymienionych czytanie zajmuje więcej czasu. Ponadto istotna jest struktura. Z tego powodu nie zawsze najlepszym wariantem jest czytanie od góry do dołu oraz od lewej do prawej strony. I wreszcie, istotne są szczegóły. Niewielkie błędy pisowni i interpunkcji, z którymi można sobie poradzić w przypadku języków naturalnych, w języku formalnym mogą mieć decydujące znaczenie.

Debugowanie

Programiści popełniają błędy. Z dziwnych powodów błędy pojawiające się w czasie programowania są potocznie nazywane **pluskwami** (ang. *bugs*), a proces ich wychwytywania to **debugowanie**.

Programowanie, a zwłaszcza debugowanie, wywołuje czasami silne emocje. Jeśli borykasz się z trudnym do usunięcia błędem, możesz czuć wściekłość, smutek lub zakłopotanie.

Przygotowanie się na takie reakcje może ułatwić poradzenie sobie z nimi. Jednym ze sposobów jest potraktowanie komputera jak pracownika z określonymi mocnymi stronami, takimi jak szybkość i dokładność, a także z konkretnymi słabymi stronami, takimi jak brak empatii i niezdolność myślenia całościowego.

Twoim zadaniem jest zostać dobrym menedżerem: znajdź sposoby na wykorzystanie mocnych stron i zminimalizowanie tych słabych. Określ również sposoby użycia własnych emocji do zaangażowania się w problem bez ryzyka, że Twoje reakcje będą uniemożliwiać efektywną pracę.

Uczenie się debugowania może być frustrujące, lecz debugowanie jest wartościową umiejętnością, która okazuje się przydatna w przypadku wielu działań wykraczających poza programowanie. Na końcu każdego rozdziału znajduje się podrozdział taki jak ten, w którym znajdziesz moje sugestie dotyczące debugowania. Mam nadzieję, że będą pomocne!

Słownik

operator arytmetyczny

Symbole, takie jak + i *, które oznaczają operacje arytmetyczne (np. dodawanie lub mnożenie).

liczba całkowita

Typ reprezentujący liczby całkowite.

liczba zmiennoprzecinkowa

Typ reprezentujący liczby z częściami ułamkowymi.

dzielenie całkowitoliczbowe

Operator (//), który umożliwia podzielenie dwóch liczb i zaokrąglenie w dół do liczby całkowitej.

wyrażenie

Kombinacja zmiennych, wartości i operatorów.

wartość

Liczba całkowita, liczba zmiennoprzecinkowa lub łańcuch albo jeden z innych rodzajów wartości, o których będzie mowa później.

funkcja

Nazwana sekwencja instrukcji wykonujących użyteczną operację. Funkcje mogą, ale nie muszą, pobierać argumenty, a ponadto ewentualnie mogą zwracać wynik.

wywołanie funkcji

Wyrażenie lub jego część, które uruchamia funkcję. Wywołanie składa się z nazwy funkcji, po której następuje lista argumentów w nawiasach okrągłych.

błąd składni

Błąd w programie, który uniemożliwia jego analizę składniową, czyli nie pozwala na jego uruchomienie.

łańcuch

Typ reprezentujący ciąg znaków.

konkatenacja

Łączenie dwóch całych łańcuchów.

typ

Kategoria wartości. Dotychczas zaprezentowane typy to liczby całkowite (typ `int`), liczby zmiennoprzecinkowe (typ `float`) i łańcuchy (typ `str`).

argument

Jedna z wartości przetwarzanych przez operator.

język naturalny

Dowolny naturalnie rozwinięty język, jakim mówią ludzie.

język formalny

Dowolny język stworzony przez ludzi do konkretnych celów, takich jak przedstawianie koncepcji matematycznych lub reprezentowanie programów komputerowych. Wszystkie języki programowania to języki formalne.

pluskwa

Błąd w programie.

debugowanie

Proces znajdowania i usuwania błędów.

Ćwiczenia

Zapytaj wirtualnego asystenta

Aby w trakcie lektury książki ułatwić sobie naukę, możesz na kilka sposobów skorzystać z wirtualnego asystenta lub chatbota:

- Jeśli chcesz dowiedzieć się więcej o zagadnieniu omawianym w rozdziale lub okaże się, że coś jest niejasne, możesz poprosić o objaśnienie.
- Jeżeli masz trudności z którymkolwiek z ćwiczeń, możesz poprosić o pomoc.

W każdym rozdziale zasugeruję ćwiczenia, które możesz wykonać z użyciem wirtualnego asystenta. Zachęcam Cię jednak do samodzielnego wypróbowania różnych rzeczy i sprawdzenia, co działa w Twoim przypadku.

Oto kilka zagadnień, o które możesz zapytać wirtualnego asystenta:

- Wcześniej wspomniałem o operatorach bitowych, ale nie wyjaśniłem, dlaczego wartość operacji `7 ^ 2` to `5`. Spróbuj zadać następujące pytania: „Jakie są operatory bitowe w języku Python?” lub „Jaka jest wartość operacji `7 XOR 2`?”.
- Wspomniałem również o kolejności operacji. Aby uzyskać więcej szczegółów, zadaj następujące pytanie: „Jaka jest kolejność operacji w języku Python?”.
- Funkcja `round`, której użyto do zaokrąglenia liczby zmiennoprzecinkowej do najbliższej liczby całkowitej, może pobrać drugi argument. Spróbuj zadać następujące pytania: „Jakie są argumenty funkcji `round`?” lub „Jak zaokrąglić liczbę pi do trzech miejsc po przecinku?”.
- Jest jeszcze jeden operator arytmetyczny, którego nie wymieniłem. Spróbuj sformułować następujące pytanie: „Jaki jest operator modulo w języku Python?”.

Większość wirtualnych asystentów wie o istnieniu języka Python, dlatego odpowiadają na takie pytania dość niezawodnie. Pamiętaj jednak o tym, że narzędzia te mogą popełniać błędy. Jeśli otrzymasz kod od chatbota, przetestuj go!

Ćwiczenie

Możesz się zastanawiać, co zrobi funkcja `round`, jeśli na końcu liczby znajduje się `.5`. Odpowiedź jest taka, że czasami funkcja ta zaokrągla w górę, a czasami w dół. Wypróbuj ten przykład, aby się przekonać o tym, czy możesz ustalić, jaką zasadą się ona kieruje:

```
round(42.5)
42
round(43.5)
44
```

Jeśli jesteś ciekawy, zadaj wirtualnemu asystentowi następujące pytanie: „Czy jeśli na końcu liczby znajduje się `.5`, w języku Python ma miejsce zaokrąglanie w górę, czy w dół?”.

Ćwiczenie

Gdy zaznajamiasz się z nowym elementem, powinieneś go wypróbować i celowo popełnić błędy. Dzięki temu dowiesz się, jakie jest znaczenie komunikatów o błędzie, a gdy ujrzysz je ponownie, będziesz wiedzieć, na co one wskazują. Lepiej popełnić błędy teraz i świadomie niż później i przypadkowo.

1. Znaku minus możesz użyć do określenia liczby ujemnej (np. `-2`). Co będzie, gdy przed liczbą wstawisz znak plus? A co będzie w przypadku `2++2`?
2. Co się dzieje, gdy występują dwie wartości bez żadnego operatora między nimi (np. `4 2`)?
3. Jeśli wywołasz funkcję, taką jak `round(42.5)`, co się stanie w przypadku pominięcia jednego lub obu nawiasów?

Ćwiczenie

Jak pamiętasz, każde wyrażenie ma wartość, każda wartość posiada typ, a my możemy użyć funkcji `type`, aby dowiedzieć się, jaki jest typ dowolnej wartości.

Jaki jest typ wartości poniższych wyrażeń? Postaraj się jak najlepiej odgadnąć typ dla każdego z nich, a następnie użyj funkcji `type`, aby to potwierdzić.

- `765`
- `2.718`
- `'2 pi'`
- `abs(-7)`
- `abs(-7.0)`
- `abs`
- `int`
- `type`

Ćwiczenie

Następujące pytania dają Ci szansę praktycznego sprawdzenia się w zapisywaniu wyrażeń arytmetycznych:

1. Ile łącznie sekund jest w 42 minutach i 42 sekundach?
2. Ile mil mieści się w 10 kilometrach? Wskazówka: jednej mili odpowiada 1,61 kilometra.

3. Jeśli 10-kilometrowy dystans wyścigu pokonasz w czasie 42 minut i 42 sekund, jakie będzie Twoje średnie tempo jako czas przypadający na milę wyrażony w sekundach?
4. Jakie jest Twoje średnie tempo jako czas przypadający na milę wyrażony w minutach i sekundach?
5. Jaka jest Twoja średnia prędkość w milach na godzinę?

Jeśli już znasz zmienne, możesz ich użyć w tym ćwiczeniu. W przeciwnym razie możesz wykonać ćwiczenie bez nich, a poznasz je już w następnym rozdziale.

Wyjątkowy przewodnik dla osób zainteresowanych nauką programowania od podstaw!

Luciano Ramalho, autor książki *Zaawansowany Python*

Python to wspaniały język programowania. Jest wszechstronny, wyrazisty i zwięzły, pozwala też korzystać z rosnącej kolekcji narzędzi i bibliotek. Cenią go zarówno profesjonalni twórcy oprogramowania, jak i amatorzy czy osoby spoza branży, które w Pythonie widzą cenne narzędzie do tworzenia własnych aplikacji, znacząco poprawiających jakość i wydajność pracy.

To trzecie wydanie przejrzystego przewodnika, który ułatwi Ci naukę programowania w Pythonie. Zaczniesz od przyswojenia podstawowych pojęć programistycznych, aby wkrótce płynnie posługiwać się funkcjami i strukturami danych. Zdobędziesz też umiejętność programowania zorientowanego obiektowo. W tym zaktualizowanym wydaniu znajdziesz również wskazówki, dzięki którym zastosujesz duże modele językowe, takie jak ChatGPT, do nauki programowania. Dowiesz się, jak tworzyć skuteczne zapytania dla tych modeli, a także jak testować i debugować kod Pythona. Dzięki ćwiczeniom, zamieszczonym w każdym rozdziale, będziesz stopniowo szlifować umiejętności programistyczne, a zasugerowane w książce strategie pomogą Ci w unikaniu frustrujących błędów — w ten sposób szybko nauczysz się tworzyć poprawny kod.

W książce:

- podstawy Pythona
- zmienne, instrukcje, funkcje i struktury danych
- praca z plikami i bazami danych
- obiekty, metody i programowanie zorientowane obiektowo
- obsługa błędów składniowych, wykonawczych i semantycznych
- użycie dużych modeli językowych do przyspieszenia nauki programowania

Allen B. Downey jest emerytowanym profesorem Olin College of Engineering. Obecnie pełni funkcję głównego data scientist w PyMC Labs. Jest znany jako autor serii podręczników z zakresu informatyki i analizy danych.

Dzięki tej książce nauczysz się używać dużych modeli językowych do nauki programowania!

Sam Lau, współautor książki *Learning Data Science*

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1902-0
9 788328 919020
Cena: 74,90 zł