

O'REILLY®

Wydanie II



Myśl w języku Python!

Helion 

Allen B. Downey

Tytuł oryginału: Think Python: How to Think Like a Computer Scientist, 2nd Edition

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-3002-3

© 2017 Helion S.A.

Authorized Polish translation of the English edition of Think Python, 2E ISBN 9781491939369

© 2016 Allen Downey

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/myjep2.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/myjep2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Przedmowa	11
1. Jak w programie	21
Czym jest program?	21
Uruchamianie interpretera języka Python	22
Pierwszy program	23
Operatory arytmetyczne	23
Wartości i typy	24
Języki formalne i naturalne	25
Debugowanie	26
Słownik	27
Ćwiczenia	29
2. Zmienne, wyrażenia i instrukcje	31
Instrukcje przypisania	31
Nazwy zmiennych	31
Wyrażenia i instrukcje	32
Tryb skryptowy	33
Kolejność operacji	34
Operacje na łańcuchach	35
Komentarze	35
Debugowanie	36
Słownik	36
Ćwiczenia	38
3. Funkcje	39
Wywołania funkcji	39
Funkcje matematyczne	40
Złożenie	41
Dodawanie nowych funkcji	41

Definicje i zastosowania	42
Przepływ wykonywania	43
Parametry i argumenty	43
Zmienne i parametry są lokalne	44
Diagramy stosu	45
Funkcje „owocne” i „puste”	46
Dlaczego funkcje?	47
Debugowanie	47
Słownik	48
Ćwiczenia	49
4. Analiza przypadku: projekt interfejsu	53
Moduł turtle	53
Proste powtarzanie	54
Ćwiczenia	55
Hermetyzowanie	56
Uogólnianie	56
Projekt interfejsu	57
Refaktoryzacja	58
Plan projektowania	59
Notka dokumentacyjna	60
Debugowanie	60
Słownik	61
Ćwiczenia	62
5. Instrukcje warunkowe i rekurencja	65
Dzielenie bez reszty i wartość bezwzględna	65
Wyrażenia boolowskie	66
Operatory logiczne	66
Wykonywanie warunkowe	67
Wykonywanie alternatywne	67
Łączuchowe instrukcje warunkowe	68
Zagnieżdżone instrukcje warunkowe	68
Rekurencja	69
Diagramy stosu dla funkcji rekurencyjnych	70
Rekurencja nieskończona	71
Dane wprowadzane z klawiatury	71
Debugowanie	72
Słownik	73
Ćwiczenia	74

6. Funkcje „owocne”	79
Wartości zwracane	79
Projektowanie przyrostowe	80
Złożenie	82
Funkcje boolowskie	82
Jeszcze więcej rekurencji	83
„Skok wiary”	85
Jeszcze jeden przykład	86
Sprawdzanie typów	86
Debugowanie	87
Słownik	88
Ćwiczenia	89
7. Iteracja	91
Ponowne przypisanie	91
Aktualizowanie zmiennych	92
Instrukcja while	92
Instrukcja break	94
Pierwiastki kwadratowe	94
Algorytmy	96
Debugowanie	96
Słownik	97
Ćwiczenia	98
8. Łańcuchy	101
Łańcuch jest ciągiem	101
Funkcja len	102
Operacja przechodzenia za pomocą pętli for	102
Fragmenty łańcuchów	103
Łańcuchy są niezmiennie	104
Wyszukiwanie	104
Wykonywanie pętli i liczenie	105
Metody łańcuchowe	105
Operator in	106
Porównanie łańcuchów	107
Debugowanie	107
Słownik	109
Ćwiczenia	110

9. Analiza przypadku: gra słów	113
Odczytywanie list słów	113
Ćwiczenia	114
Wyszukiwanie	115
Wykonywanie pętli z wykorzystaniem indeksów	116
Debugowanie	117
Słownik	118
Ćwiczenia	118
10. Listy	121
Lista to ciąg	121
Listy są zmienne	122
Operacja przechodzenia listy	123
Operacje na listach	123
Fragmety listy	124
Metody list	124
Odwzorowywanie, filtrowanie i redukowanie	125
Usuwanie elementów	126
Listy i łańcuchy	127
Obiekty i wartości	127
Tworzenie aliasu	128
Argumenty listy	129
Debugowanie	131
Słownik	132
Ćwiczenia	133
11. Słowniki	137
Słownik to odwzorowanie	137
Słownik jako kolekcja liczników	139
Wykonywanie pętli i słowniki	140
Wyszukiwanie odwrotne	140
Słowniki i listy	141
Wartości zapamiętywane	143
Zmienne globalne	144
Debugowanie	146
Słownik	146
Ćwiczenia	148

12. Krotki	151
Krotki są niezmiennie	151
Przypisywanie krotki	152
Krotki jako wartości zwracane	153
Krotki argumentów o zmiennej długości	153
Listy i krotki	154
Słowniki i krotki	156
Ciągi ciągów	157
Debugowanie	158
Słownik	159
Ćwiczenia	159
13. Analiza przypadku: wybór struktury danych	163
Analiza częstości występowania słów	163
Liczby losowe	164
Histogram słów	165
Najczęściej używane słowa	166
Parametry opcjonalne	167
Odejmnowanie słowników	167
Słowa losowe	168
Analiza Markowa	169
Struktury danych	171
Debugowanie	172
Słownik	173
Ćwiczenia	174
14. Pliki	175
Trwałość	175
Odczytywanie i zapisywanie	175
Operator formatu	176
Nazwy plików i ścieżki	177
Przechwytywanie wyjątków	178
Bazy danych	179
Użycie modułu pickle	180
Potoki	181
Zapisywanie modułów	182
Debugowanie	183
Słownik	183
Ćwiczenia	184

15. Klasy i obiekty	187
Typy definiowane przez programistę	187
Atrybuty	188
Prostokąty	189
Instancje jako wartości zwracane	190
Obiekty są zmienne	190
Kopiowanie	191
Debugowanie	192
Słownik	193
Ćwiczenia	194
16. Klasy i funkcje	195
Klasa Time	195
Funkcje „czyste”	196
Modyfikatory	197
Porównanie prototypowania i planowania	198
Debugowanie	199
Słownik	200
Ćwiczenia	201
17. Klasy i metody	203
Elementy obiektowe	203
Wyświetlanie obiektów	204
Kolejny przykład	205
Bardziej złożony przykład	206
Metoda init	206
Metoda __str__	207
Przeciążanie operatorów	207
Przekazywanie oparte na typie	208
Polimorfizm	209
Interfejs i implementacja	210
Debugowanie	211
Słownik	211
Ćwiczenia	212
18. Dziedziczenie	213
Obiekty kart	213
Atrybuty klasy	214
Porównywanie kart	215
Talie	216
Wyświetlanie talii	216

Dodawanie, usuwanie, przenoszenie i sortowanie	217
Dziedziczenie	218
Diagramy klas	219
Hermetyzacja danych	220
Debugowanie	221
Słownik	222
Ćwiczenia	223
19. Przydatne elementy	227
Wyrażenia warunkowe	227
Wyrażenia listowe	228
Wyrażenia generatora	229
Funkcje any i all	230
Zbiory	230
Liczniki	232
defaultdict	232
Krotki z nazwą	234
Zbieranie argumentów słów kluczowych	235
Słownik	236
Ćwiczenia	236
20. Debugowanie	237
Błędy składniowe	237
Błędy uruchomieniowe	239
Błędy semantyczne	242
21. Analiza algorytmów	247
Tempo wzrostu	248
Analiza podstawowych operacji w języku Python	250
Analiza algorytmów wyszukiwania	252
Tablice mieszające	252
Słownik	256
Skorowidz	257

Jak w programie

Celem tej książki jest nauczenie Cię 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.

W przypadku informatyka najważniejszą pojedynczą umiejętnością jest **rozwiązywanie problemów**. Oznacza to zdolność formułowania problemów, kreatywnego myślenia o problemach i przedstawiania ich w dokładny i przejrzysty sposób. Jak się okazuje, proces uczenia programowania to znakomita sposobność do sprawdzenia umiejętności rozwiązywania problemów. Z tego właśnie powodu ten rozdział nosi tytuł „Jak w programie”.

Na jednym poziomie będziesz uczyć się programowania, które samo w sobie jest przydatną umiejętnością. Na innym wykorzystasz programowanie jako środek do osiągnięcia celu. W trakcie lektury kolejnych rozdziałów cel ten stanie się bardziej wyraźny.

Czym jest program?

Program to sekwencja instrukcji określających, w jaki sposób ma zostać przeprowadzone obliczenie. Obliczenie może mieć postać jakiegoś działania matematycznego, tak jak w przypadku rozwiązywania układu równań lub znajdowania pierwiastków wielomianu, ale może też być obliczeniem symbolicznym (przykładem jest wyszukiwanie i zastępowanie tekstu w dokumencie) lub czymś w postaci operacji graficznej (jak przetwarzanie obrazu lub odtwarzanie wideo).

Szczegóły prezentują się inaczej w różnych językach, ale kilka podstawowych elementów pojawia się w niemal każdym języku. Oto one:

dane wejściowe

Dane wprowadzone za pomocą klawiatury albo pochodzące z pliku, sieci lub jakiegoś urządzenia.

dane wyjściowe

Dane wyświetlane na ekranie, zapisywane w pliku, wysyłane za pośrednictwem sieci itp.

działania matematyczne

Podstawowe operacje matematyczne, takie jak dodawanie i mnożenie.

wykonywanie warunkowe

Sprawdzanie określonych warunków i uruchamianie odpowiedniego kodu.

powtarzanie

Wielokrotne wykonywanie pewnego działania (zwykle zmieniającego się w pewien sposób).

Czy temu wierzyć, czy nie, to naprawdę wszystko, co jest związane z programem. Każdy program, jakiego dotąd używałeś, nieważne jak bardzo skomplikowany, tak naprawdę jest złożony z elementów podobnych do wyżej wymienionych. Oznacza to, że programowanie możesz postrzegać jako proces dzielenia dużego i złożonego zadania na coraz mniejsze podzadania do momentu, aż są one na tyle proste, że sprowadzają się do jednego z powyższych podstawowych elementów.

Uruchamianie interpretera języka Python

Jednym z wyzwań przy rozpoczynaniu przygody z językiem Python jest ewentualna konieczność instalacji na komputerze tego języka wraz z powiązanim oprogramowaniem. Jeśli jesteś zaznajomiony ze swoim systemem operacyjnym, a zwłaszcza z interfejsem wiersza poleceń, nie będziesz mieć problemu z instalacją języka Python. Dla początkujących utrudnieniem może być jednak konieczność równoczesnego przyswajania wiedzy z zakresu administrowania systemem i programowania.

Aby uniknąć tego problemu, zalecam na początek uruchomienie interpretera języka Python w przeglądarce. Gdy będziesz zaznajomiony z tym językiem, zaprezentuję sugestie dotyczące instalowania go na komputerze.

Dostępnych jest kilka stron internetowych umożliwiających uruchomienie interpretera języka Python. Jeśli masz już swojego faworyta, po prostu z niego skorzystaj. W przeciwnym razie polecam witrynę PythonAnywhere. Pod adresem <http://tinyurl.com/thinkpython2e> zamieszczono szczegółowe instrukcje pozwalające na rozpoczęcie pracy.

Istnieją dwie wersje języka Python, czyli Python 2 i Python 3. Ponieważ są one bardzo podobne, po poznaniu jednej z nich z łatwością można zacząć korzystać z drugiej. Okazuje się, że występuje tylko kilka różnic, z jakimi będziesz mieć do czynienia jako początkujący. Tę książkę napisano z myślą o języku Python 3, ale uwzględniono kilka uwag dotyczących języka Python 2.

Interpreter języka Python to program odczytujący i wykonujący kod Python. Zależnie od używanego środowiska w celu uruchomienia interpretera może być wymagane kliknięcie ikony lub wpisanie polecenia `python` w wierszu poleceń. Po uruchomieniu interpretera powinny być widoczne dane wyjściowe podobne do następujących:

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Pierwsze trzy wiersze zawierają informacje dotyczące interpretera i systemu operacyjnego, w którym go uruchomiono, dlatego możesz ujrzeć coś innego. Należy jednak sprawdzić, czy numer wersji,

który w przykładzie ma postać 3.4.0, rozpoczyna się od cyfry 3 wskazującej, że uruchomiono interpreter języka Python 3. Jeśli numer wersji zaczyna się cyfrą 2, załadowano interpreter (pewnie się domyśliłeś) języka Python 2.

Ostatni wiersz to **wiersz zachęty** wskazujący, że interpreter jest gotowy do przyjęcia kodu wprowadzonego przez użytkownika. Jeśli wpiszesz wiersz kodu i naciśniesz klawisz *Enter*, interpreter wyświetli następujący wynik:

```
>>> 1 + 1
2
```

Możesz teraz przejść do dzieła. Od tego momentu zakładam, że wiesz, jak załadować interpreter języka Python i uruchomić kod.

Pierwszy program

Tradycyjnie pierwszy program, jaki piszesz w nowym języku, nosi nazwę *Witaj, świecie!*, ponieważ wyświetla on właśnie te słowa: *Witaj, świecie!*. W języku Python wygląda to następująco:

```
>>> print('Witaj, świecie!')
```

Jest to przykład **instrukcji print**, choć w rzeczywistości nie powoduje ona drukowania niczego na papierze. Instrukcja wyświetla wynik na ekranie. W tym przypadku wynikiem są następujące słowa:

```
Witaj, świecie!
```

Znaki pojedynczego cudzysłowu w kodzie programu oznaczają początek i koniec tekstu do wyświetlenia. Te znaki nie pojawiają się w wyniku.

Nawiasy okrągłe wskazują, że instrukcja `print` to funkcja. Funkcjami zajmiemy się w rozdziale 3.

W języku Python 2 instrukcja `print` jest trochę inna. Ponieważ nie jest funkcją, nie korzysta z nawiasów okrągłych.

```
>>> print 'Witaj, świecie!'
```

To rozróżnienie nabierze wkrótce większego sensu, ale na początek tyle wystarczy.

Operatory arytmetyczne

Po programie *Witaj, świecie!* następny krok to arytmetyka. Język Python zapewnia **operatory**, które są specjalnymi symbolami reprezentującymi takie obliczenia jak dodawanie i mnożenie.

Operatory `+`, `-` i `*` służą do wykonywania dodawania, odejmowania i mnożenia, tak jak w następujących przykładach:

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42
```

Operator / wykonuje operację dzielenia:

```
>>> 84 / 2
42.0
```

Możesz się zastanawiać, dlaczego wynik to 42.0, a nie 42. Zostanie to wyjaśnione w następnym podrozdziale.

I wreszcie, operator ** służy do potęgowania, czyli podniesienia liczby do potęgi:

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

W niektórych innych językach na potrzeby potęgowania używany jest operator ^, ale w języku Python jest to operator bitowy o nazwie XOR. Jeśli nie jesteś zaznajomiony z operatorami bitowymi, następujący wynik zaskoczy Cię:

```
>>> 6^2
4
```

W tej książce nie są omawiane operatory bitowe, ale możesz o nich poczytać na stronie dostępnej pod adresem <http://wiki.python.org/moin/BitwiseOperators>.

Wartości i typy

Wartość to jeden z podstawowych elementów używanych przez program, jak litera lub liczba. Niektóre dotychczas zaprezentowane wartości to 2, 42.0 oraz Witaj, świecie!.

Wartości te należą do różnych **typów**: liczba 2 to **liczba całkowita**, 42.0 to **liczba zmiennoprzecinkowa**, a Witaj, świecie! to **łańcuch** (taka nazwa wynika z tego, że litery tworzą jedną całość).

Jeśli nie masz pewności, jakiego typu jest wartość, interpreter może zapewnić taką informację:

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Witaj, świecie!')
<class 'str'>
```

W powyższych wynikach słowo class odgrywa rolę kategorii. Typ to kategoria wartości.

Nie jest zaskoczeniem to, że liczby całkowite należą do typu int, łańcuchy do typu str, a liczby zmiennoprzecinkowe do typu float.

A co z wartościami takimi jak '2' i '42.0'? Wyglądają one jak liczby, ale ujęto je w znaki cudzo-słowa, tak jak łańcuchy:

```
>>> type('2')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

Są to łańcuchy.

Gdy w krajach anglojęzycznych używana jest duża liczba całkowita, jej grupy cyfr są oddzielane przecinkiem (np. 1,000,000). Choć tak zapisana liczba jest poprawna, w języku Python jest niedozwolona *liczbą całkowitą*:

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

Czegoś takiego zupełnie nie oczekujemy! W języku Python liczba 1,000,000 interpretowana jest jako sekwencja liczb całkowitych oddzielonych przecinkiem. W dalszej części rozdziału dowiesz się więcej o tego rodzaju sekwencji.

Języki formalne i naturalne

Języki naturalne to języki, jakimi posługują się ludzie, takie jak angielski, hiszpański i francuski. Nie zostały stworzone przez ludzi (choć ludzie próbują narzucać w nich jakiś porządek), 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. Chemicy używają języka formalnego do reprezentowania struktury chemicznej molekuł. I co najważniejsze:

Języki programowania to języki formalne, które zostały zaprojektowane do definiowania obliczeń.

Języki formalne mają zwykle ściśle reguły dotyczące **składni**, które decydują o strukturze instrukcji. Na przykład w matematyce równanie $3+3 = 6$ ma poprawną składnię, ale wyrażenie $3+ = 3\$6$ już nie. W chemii H_2O to poprawny składniowo wzór, ale w przypadku $_2Zz$ tak nie jest.

Występują dwie odmiany reguł dotyczących składni. Pierwsza odmiana związana jest z **tokenami**, a druga ze strukturą. Tokeny to podstawowe elementy języka, takie jak słowa, liczby i symbole chemiczne. Jednym z problemów w przypadku wyrażenia $3+ = 3\$6$ jest to, że znak \$ nie jest w matematyce uznawany za poprawny token (tak przynajmniej mi wiadomo). Podobnie wzór $_2Zz$ nie jest dozwolony, ponieważ nie istnieje element ze skrótem Zz .

Drugi typ reguły dotyczącej składni powiązany jest ze sposobem łączenia tokenów. Równanie $3+ = 3$ jest niepoprawne, ponieważ nawet pomimo tego, że $+ i =$ to poprawne tokeny, niedozwolona jest sytuacja, gdy jeden następuje bezpośrednio po drugim. Podobnie we wzorze chemicznym indeks dolny musi znajdować się po nazwie elementu, a nie przed nią.

To jest zdanie w języku polskim @ poprawnej strukturze, które zawiera niewłaściwe tokeny. Z kolei zdanie to wszystkie tokeny poprawne ma, nieprawidłową ale strukturę.

Gdy czytasz zdanie w języku polskim lub instrukcję w języku formalnym, musisz określić strukturę (w języku naturalnym robisz to podświadomie). Proces ten nazywany jest **analizą składni**.

Chociaż języki formalne i naturalne mają wiele wspólnych elementów, takich jak tokeny, struktura i składnia, występują pomiędzy nimi także następujące 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 każda instrukcja, niezależnie od kontekstu, ma dokładnie jedno znaczenie.

nadmiarowość

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

dosłowność

Języki naturalne są pełne idiomów i metafor. Jeśli ktoś powie „Mleko się rozlało”, nie oznacza to raczej, że gdzieś naprawdę rozlało się mleko (idiom ten znaczy, że wydarzyło się coś, czego nie można już cofnąć). 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. Różnica między językiem formalnym i naturalnym jest taka jak między poezją i prozą, tym bardziej że:

Poezja

W przypadku słów istotne jest zarówno ich brzmienie, jak i znaczenie. Cały wiersz tworzy efekt lub reakcję emocjonalną. Wieloznaczność jest nie tylko typowa, ale często zamierzona.

Proza

Ważniejsze jest dosłowne znaczenie słów, a struktura zawiera w sobie więcej znaczenia. Proza jest łatwiejsza do analizy niż poezja, ale i ona często cechuje się wieloznacznością.

Programy

Znaczenie programu komputerowego jest jednoznaczne i dosłowne. Program może zostać całkowicie zrozumiany w wyniku analizy tokenów i struktury.

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. Zamiast tego naucz się analizować program w głowie, identyfikując tokeny i interpretując strukturę. 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 **pluskami** (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, zniechęcenie lub zakłopotanie.

Świadczy to o tym, że ludzie w naturalny sposób odpowiadają komputerom tak, jakby były ludźmi. Gdy działają dobrze, traktujemy je jak kolegów z zespołu, a gdy okazują się „zawzięte” lub „nie-miłe”, reagujemy tak samo jak w przypadku upartych i nieomyłych osób (Reeves i Nass, *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*).

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

rozwiązywanie problemu

Proces formułowania problemu, znajdowania rozwiązania i wyrażania go.

język wysokiego poziomu

Język programowania, taki jak Python, zaprojektowany tak, aby pozwalał ludziom w prosty sposób pisać i czytać kod.

język niskiego poziomu

Język programowania zaprojektowany tak, aby jego kod z łatwością mógł zostać uruchomiony przez komputer. Język ten nazywany jest również językiem maszynowym lub językiem assemblera.

przenośność

Właściwość programu umożliwiająca uruchomienie go na więcej niż jednym typie komputera.

interpreter

Program wczytujący inny program i wykonujący go.

zachęta

Znaki wyświetlane przez interpreter w celu wskazania, że jest on gotowy do pobrania danych wejściowych od użytkownika.

program

Zestaw instrukcji określających obliczenie.

instrukcja wyświetlająca

Instrukcja powodująca wyświetlenie przez interpreter języka Python wartości na ekranie.

operator

Specjalny symbol reprezentujący prostą operację, taką jak dodawanie, mnożenie lub łączenie łańcuchów.

wartość

Jedna z podstawowych jednostek danych, takich jak liczba lub łańcuch, która jest przetwarzana przez program.

typ

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

liczba całkowita

Typ reprezentujący liczby całkowite.

liczba zmiennoprzecinkowa

Typ reprezentujący liczby z częściami ułamkowymi.

łańcuch

Typ reprezentujący sekwencje znaków.

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.

token

Jeden z podstawowych elementów struktury składniowej programu analogiczny do słowa w języku naturalnym.

składnia

Reguły zarządzające strukturą programu.

analiza składni

Ma na celu sprawdzenie programu i dokonanie analizy struktury składniowej.

pluskwa

Błąd w programie.

Proces znajdowania i usuwania błędów.

Ćwiczenia

Ćwiczenie 1.1.

Dobrym pomysłem będzie czytanie tej książki przed komputerem, aby mieć możliwość sprawdzania przykładów na bieżąco.

Każdorazowo, gdy eksperymentujesz z nowym elementem, spróbuj popełnić błędy. Co będzie, gdy na przykład w programie *Witaj, świecie!* zostanie pominięty jeden ze znaków cudzysłowu? Co się stanie, jeśli pominiesz oba te znaki? Co będzie, gdy niepoprawnie wprowadzisz nazwę instrukcji `print`?

Tego rodzaju eksperyment ułatwia zapamiętanie tego, co czytasz. Pomocny jest również podczas programowania, ponieważ postępując w ten sposób, poznajesz znaczenie komunikatów o błędzie. Lepiej popełnić błędy teraz i świadomie niż później i przypadkowo.

1. Co się stanie, gdy w instrukcji `print` zostanie pominięty jeden z nawiasów okrągłych lub oba?
2. Jeśli próbujesz wyświetlić łańcuch, co się stanie, gdy pominiesz jeden ze znaków cudzysłowu lub oba?
3. 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`?
4. W zapisie matematycznym zera umieszczone na początku są poprawne (np. `02`). Co się stanie, jeśli czegoś takiego spróbujesz w przypadku języka Python?
5. Co się dzieje, gdy występują dwie wartości bez żadnego operatora między nimi?

Ćwiczenie 1.2.

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 (czas przypadający na milę wyrażony w minutach i sekundach)? Jaka jest Twoja średnia prędkość w milach na godzinę?

A

aktualizacja, 97
aktualizowanie zmiennych, 92
akumulator, 132
algorytm, 96, 247
 kwadratowy, 257
 liniowy, 257
 wyszukiwania, 252
alias, 128
analiza
 algorytmów, 247, 256
 algorytmów wyszukiwania, 252
 częstości, 163
 Markowa, 169
 operacji, 250
 porównawcza, 174
 przypadku
 gra słów, 113
 wybór struktury danych, 163
 składni, 25, 28
argument, 43
 funkcji, 37, 39, 48
 listy, 129
 opcjonalny, 110
 pozycyjny, 206, 211
 słowa kluczowego, 61, 235
atrybut, 188, 193
 klasy, 214, 222

B

bazy danych, 179, 184
błąd
 AttributeError, 241
 IndexError, 242
 KeyError, 241
 TypeError, 241

błędy

 kształtu, 158
 semantyczne, 38, 237, 242
 składniowe, 36, 38, 237
 uruchomieniowe, 36, 237, 239

C

ciągi, 101, 109, 121
 ciągów, 157
 formatu, 184
częstość, 163
 używania słów, 166

D

dane
 wprowadzane z klawiatury, 71
 wyjściowe, 242
debugowanie, 26, 36, 60, 87, 107, 146, 192, 237
 z użyciem gumowej kaczuszki, 174
definicja funkcji, 41, 48
deklaracja, 148
dekrementacja, 97
diagram
 klas, 219, 223
 obiektów, 188, 194, 215
 stanu, 31, 37, 142, 157
 stosu, 45, 49
 dla funkcji rekurencyjnych, 70
dodawanie
 kart, 217
 nowych funkcji, 41
dziedziczenie, 213, 218, 223
dzielenie bez reszty, 65, 73

E

elementy, 109, 121, 132, 147
obiektywne, 203

F

fabryka, 233, 236
filtrowanie, 125, 126, 132
flaga, 144, 148
formatowanie danych wyjściowych, 146
fragment, 109
 listy, 124
 łańcucha, 103
funkcja, 39, 48, 195
 „owocna”, 48
 „pusta”, 49
 all, 230
 any, 230
 arc, 58
 avoids, 115
 dict, 156
 float, 39
 len, 102, 138
 os.path.isdir, 178
 os.path.join, 178
 print_attributes, 211
 randint, 164
 random, 164
 str, 40
 uses_all, 116
 uses_only, 115
 zip, 156
funkcje
 boolowskie, 82
 czyste, 196, 200
 kwadratowe, 249
 matematyczne, 40
 mieszające, 143, 147
 owocne, 46, 79
 polimorficzne, 210
 puste, 46

G

gałąź, 74
gra słów, 113
graf wywołań, 147

H

hermetyzacja, 56, 61
 danych, 220, 223
histogram, 139
 słów, 165

I

identyczność, 133
implementacja, 139, 147, 210
indeks, 109, 116
inicjalizacja, 97
inkrementacja, 97
instancja, 188, 193
instancje jako wartości zwracane, 190
instrukcja, 32, 37
 break, 94
 import, 40, 49
 print, 23
 raise, 141, 147
 return, 70, 74, 79
 while, 92
instrukcje
 asercji, 200
 globalne, 145, 148
 przypisania, 31
 rozszerzonego, 125
 warunkowe, 65, 74
 łańcuchowe, 68
 zagnieżdżone, 68
 wyświetlające, 28
 złożone, 67, 74
interfejs, 57, 61, 210
interpreter, 22, 27
iteracja, 91, 97
iterator, 159

J

języki
 formalne, 25, 28
 naturalne, 25, 28
 niskiego poziomu, 27
 obiektywne, 211
 wysokiego poziomu, 27

K

- katalog, 184
- klasa, 187, 193, 203
 - LinearMap, 254
 - Time, 195
- klasy
 - nadrzędne, 218, 223
 - podrzędne, 218, 223
- klucz, 137, 147
- kod „martwy”, 88
- kodowanie, 222
 - rang i kolorów, 213
- kolejność operacji, 34, 37
- kolekcja liczników, 139
- komentarz, 35, 37
- komunikat o błędzie, 39
- konkatenacja łańcuchów, 35, 37
- kopiowanie
 - głębokie, 192
 - obiektu, 191
 - płytkie, 192
- koszt operacji dodawania, 256
- krotki, 151, 159
 - argumentów, 153
 - jako wartości zwracane, 153
 - z nazwą, 234

L, ł

- liczba
 - całkowita, 24, 28
 - zmiennoprzecinkowa, 24, 28
 - losowa, 164
- licznik, 105, 110, 232
- lista, 121, 127, 132, 141, 154
 - zagnieżdżona, 121, 132
- listy zmienne, 122
- łańcuch, 24, 28, 35, 101, 127
 - formatu, 176, 183
 - niezmienny, 104
- łańcuchowa instrukcja warunkowa, 68, 74

M

- metoda, 61, 203, 211
 - __init__, 254
 - __str__, 207
 - add, 253
 - find_map, 254
 - get, 253

- init, 206
- items, 156
- metody
 - list, 124
 - łańcuchowe, 105
- mieszanie, 143
- mnożość, 223
- model komputera, 256
- moduł, 40, 49
 - collections, 232
 - copy, 192
 - os.path, 178
 - pickle, 180
 - random, 164
 - turtle, 53
- modyfikator, 197, 200
- możliwość mieszania, 147

N

- nadpisywanie, 174
- nagłówek, 41, 48
- najgorszy przypadek, 256
- nazwy
 - plików, 177
 - ścieżki, 177
 - zmiennych, 31
- niezmiennik, 199, 200
- niezmienność, 109
- notacja
 - „dużego O”, 257
 - z kropką, 40, 49
- notka dokumentacyjna, 60, 61
- NPMDDO, 64

O

- obiekt, 109, 127, 132, 187
 - bajtów, 180, 184
 - defaultdict, 232
 - dict_items, 156
 - funkcji, 48
 - funkcji zip, 159
 - HashMap, 255
 - klasy, 193
 - modułu, 49
 - osadzony, 193
 - pliku, 113, 118
 - potoku, 181, 184
 - Rectangle, 192
- obiektyowy język programowania, 203

- obiekty
 - kart, 213
 - zmienne, 190
- odczytywanie list słów, 113
- odejmowanie słowników, 167
- odwołanie, 129, 133
- odwzorowanie, 125, 137, 146
- okleina, 223
- operacje
 - na listach, 123
 - na łańcuchach, 35
 - przechodzenia, 102, 123
- operator, 28
 - in, 106, 230
- operatory
 - arytmetyczne, 23
 - formatu, 176, 183
 - logiczne, 66, 73
 - relacyjne, 66, 73
 - wartości bezwzględnej, 65, 73

P

- para klucz-wartość, 137, 147
- parametry, 43, 48
 - lokalne, 44
 - opcjonalne, 167
- pętla, 55, 61, 140
 - for, 102
- pętle nieskończone, 93, 97, 240
- pierwiastki kwadratowe, 94
- pierwszy program, 23
- plan projektowania, 59, 61
- planowanie, 198
- pliki, 175
 - nazwy, 177
 - odczytywanie, 175
 - tekstowe, 184
 - zapisywanie, 175
- pluskwa, 26, 28
- podmiot, 205, 211
- polimorfizm, 209, 212
- ponowne przypisanie, 91, 97
- porównanie
 - kart, 215
 - łańcuchów, 107
 - prototypowania i planowania, 198
- potoki, 181
- powłoka, 184
- pozycja, 121
- program, 21, 28

- programowanie
 - funkcyjne, 200
 - obiektowe, 211
- projekt interfejsu, 53, 57
- projektowanie, 59
 - przyrostowe, 80, 88
 - zaplanowane, 200
- proste powtarzanie, 54
- prostokąty, 189
- prototyp i poprawki, 200
- prototypowanie, 198
- przechodzenie, 102, 109
 - listy, 123
- przechwytywanie wyjątków, 178, 184
- przeciążanie operatorów, 207, 212
- przekazywanie oparte na typie, 208, 212
- przenoszenie kart, 217
- przenośność, 27
- przepływ wykonywania, 43, 49, 240
- przypadek bazowy, 74
- przypisanie, 37
 - krotki, 152, 159
 - rozszerzone, 125, 132
- punkt przejścia, 249, 257
- pusty łańcuch, 109

R

- ramka, 45, 49
- redukowanie, 125, 132
- refaktoryzacja, 58, 61
- rekurencja, 65, 69, 74, 83
 - nieskończona, 71, 74, 240
- relacja
 - JEST, 219, 223
 - MA, 219, 223
- rozmieszczanie, 154, 159
- rozwiązywanie problemu, 27
- równoważność, 133

S, Ś

- semantyka, 38
- separator, 127, 133
- singleton, 142, 147
- składnia, 25, 28
- składnik wiodący, 248, 256
- skok wiary, 85
- skrypt, 33, 37
- słowa losowe, 168

słownik, 137, 140, 146
słowo kluczowe, 37, 235
 class, 32
sortowanie, 217
specjalny przypadek, 118
sprawdzanie
 podsumowań, 146
 typów, 86, 146
stos, 45
strażnik, 87, 88
struktura danych, 159, 163, 171
szkielet, 81, 88
ścieżka, 177, 184
 bezwzględna, 177, 184
 względna, 177, 184
śledzenie wsteczne, 49

T

tablica mieszająca, 147, 252, 257
talie, 216
tempo wzrostu, 248, 249, 257
token, 25, 28
treść, 48
trwałość programów, 175, 183
tryb
 interaktywny, 37
 skryptowy, 33, 37
tworzenie
 aliasu, 128, 133
 automatycznych sprawdzeń, 146
 instancji, 188, 193
typy, 24, 28
 definiowane przez programistę, 187

U

ukrywanie informacji, 212
uogólnianie, 56, 61
uruchamianie interpretera, 22
usuwanie
 elementów, 126
 kart, 217
użycie
 metody gumowej kaczuszki, 172
 modułu pickle, 180

W

wartości, 24, 28, 127, 147
 listy, 121

wartość
 bezwzględna, 65
 domyślna, 173
 zapamiętywana, 143, 144, 147
 zwracana, 39, 48, 190
warunek, 74
 końcowy, 60
 wstępny, 60
wielozbiór, 232, 236
wiersz zachęty, 23
wybór struktury danych, 163
wydajność względna algorytmów, 247
wyjątek, 38, 178
wykonywanie, 37
 alternatywne, 67
 pętli, 105, 116, 140
 warunkowe, 67
wykorzystanie indeksów, 116
wyrażenia, 32, 37
 boolowskie, 66, 73
 generatora, 229, 236
 listowe, 228, 236
 warunkowe, 227, 236
wyszukiwanie, 104, 110, 115, 147, 252
 odwrotne, 140, 147
wyświetlanie
 obiektów, 204
 talii, 216
wywołanie, 106, 110
 funkcji, 39, 48
wyznaczanie wartości, 37

Z

zachęta, 27
zadeklarowanie zmiennej, 145
zagnieżdżona instrukcja warunkowa, 68, 74
zależność, 223
zapisywanie modułów, 182
zbieranie, 159
 argumentów, 153
zbiory, 230
złożenie, 41, 49, 82
zmiennie, 31, 36, 44
 globalne, 144, 148
 lokalne, 48
 tymczasowe, 79, 88
znak
 nowego wiersza, 72
 podkreślenia, 32

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ 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 WYDAWNICZA

 **Helion SA**

Python: dzięki niemu zaczniesz myśleć jak informatyk!

Aby stać się cenionym programistą, trzeba zacząć od bardzo solidnych podstaw. Python jest idealną propozycją dla osób, które chcą nauczyć się programowania. Składnia i podstawowe koncepcje programistyczne w Pythonie są dość proste do zrozumienia. Sam język ma szerokie zastosowanie w różnych dziedzinach wiedzy. Umożliwia przy tym pisanie czytelnego i łatwego w konserwacji kodu, co jest ogromną zaletą.

Trzymasz w ręku praktyczny przewodnik do nauki programowania. Znajdziesz w nim przystępnie napisane wyjaśnienia podstawowych pojęć programistycznych. Dowiesz się, jak stosować funkcje, czym jest rekurencja, jak wyglądają struktury danych i na czym polega projektowanie obiektowe. W każdym rozdziale znalazły się praktyczne ćwiczenia, dzięki którym będziesz używać poznawanych koncepcji i utrwalisz zdobytą wiedzę.

Allen B. Downey — jest profesorem informatyki na uczelni Olin College of Engineering. Prowadził zajęcia na uczelniach: Wellesley College, Colby College i U.C. Berkeley. Na tej ostatniej uzyskał tytuł doktora.

W tej książce:

- przedstawiono podstawy Pythona, w tym jego składnię i semantykę
- opisano najważniejsze koncepcje programistyczne i zdefiniowano istotne pojęcia
- pokazano, jak stosować wartości, zmienne, instrukcje, funkcje i struktury danych
- przedstawiono metody pracy z plikami i bazami danych
- wyjaśniono zagadnienia programowania obiektowego
- opisano techniki debugowania służące do usuwania błędów składniowych, uruchomieniowych i semantycznych

Helion

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne

☎ 0 801 339900

📞 0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
● <http://helion.pl/promocje>
Książki najchętniej czytane:
● <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
● <http://helion.pl/nowości>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-3002-3



9 788328 330023

Informatyka w najlepszym wydaniu

cena: 49,00 zł