O'REILLY®

Wydanie II

Myśl w języku Python!

Allen B. Downey

Helion

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

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-3002-3

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.

Spis treści

ROZDZIAŁ 1.

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 powiązanym 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 cudzysłowu, 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 niedozwoloną *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 ścisłe 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 pol$kim @ poprawnej strukturze, które zawiera niewłaściwe t*keny. 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 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. 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 **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, 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 „niemiłe", reagujemy tak samo jak w przypadku upartych i niemiłych 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 asemblera.

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.

debugowanie

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?

Skorowidz

E

F

G

H

I

J

K

L, Ł

M

N

O

P

R

S, Ś

T

U

W

Z

O'REILLY®

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.

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

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.

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/nowosci

ISBN 978-83-283-3002-3

Informatyka w najlepszym wydaniu

cena: 49,00 zł