Software Development Series

Scott Meyers, konsultant redakcyjny

Efektywny Python

125 sposobów na lepszy kod

Wydanie III

Brett Slatkin

Helion

Tytuł oryginału: Effective Python: 125 Specific Ways to Write Better Python, 3rd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-2695-0

Authorized translation from the English language edition, entitled EFFECTIVE PYTHON: 125 SPECIFIC WAYS TO WRITE BETTER PYTHON, 3rd Edition 9780138172183 by Brett Slatkin published by Pearson Education, Inc, publishing as Addison-Wesley Professional.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/efpyt3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Sposób 3. Nigdy nie oczekuj, że Python wykryje błędy podczas kompilacji

Podczas wczytywania programu i przygotowywania go do uruchomienia kod źródłowy programu zostaje przekształcony na postać drzewa składniowego i sprawdzony pod kątem oczywistych błędów strukturalnych. Na przykład kiepsko przygotowana konstrukcja `if` spowoduje zgłoszenie wyjątku `SyntaxtError`, który wskazuje na istnienie błędu w kodzie.

```
if True  # Niepoprawna składnia
  print("witaj")
```

```
>>>
Traceback ...
SyntaxError: expected ':'
```

Błędy w literałach również zostaną wychwycone na wczesnym etapie i doprowadzą do zgłoszenia odpowiednich wyjątków.

```
1.3j5  # Niepoprawna liczba
```

```
>>>
Traceback ...
SyntaxError: invalid imaginary literal
```

Niestety to wszystko w zakresie wykrywania błędów, na co można liczyć od Pythona przed uruchomieniem programu. Wszelkie nieprawidłowości wykraczające poza błędy tokenizacji i przetwarzania nie będą zgłaszane.

Nawet w przypadku prostych funkcji zawierających oczywiste błędy nie należy oczekiwać ich zgłoszenia przed uruchomieniem programu. To wiąże się z wysoce dynamiczną naturą Pythona. Na przykład spójrz na definicję funkcji, w której zmiennej `my_var` nie została przypisana wartość przed przekazaniem tej zmiennej do funkcji `print()`.

```
def bad_reference():
    print(my_var)
    my_var = 123
```

Jednak mimo to zgłoszenie wyjątku nie nastąpi aż do chwili wywołania funkcji.

```
bad_reference()
```

```
>>>
Traceback ...
UnboundLocalError: cannot access local variable 'my_var' where it is not associated
with a value
```

Tego rodzaju błąd nie jest uznawany za *statyczny*, ponieważ w programach Pythona można dynamicznie przypisywać wartości zmiennym lokalnym i globalnym. W kolejnym fragmencie kodu zamieściłem definicję funkcji, która jest poprawna i niezależna od argumentu danych wejściowych.

```
def sometimes_ok(x):
    if x:
        my_var = 123
    print(my_var)
```

Przedstawione tutaj wywołanie funkcji działa bezbłędnie.

```
sometimes_ok(True)
```

```
>>>
123
```

Natomiast kolejne prowadzi do zgłoszenia wyjątku.

```
sometimes_ok(False)
```

```
>>>
Traceback ...
UnboundLocalError: cannot access local variable 'my_var' where it is not associated
with a value
```

Python nie wychwyci z wyprzedzeniem także błędów matematycznych. Dlatego też interpreter nie będzie wiedział o istnieniu błędu w tej funkcji, dopóki nie spróbuje jej wykonać.

```
def bad_math():
    return 1 / 0
```

Sposób działania operatora dzielenia zależy od wartości operandów, więc sprawdzenie pod kątem błędów zostaje odłożone aż do chwili wykonywania tego kodu.

```
bad_math():

>>>
Traceback ...
ZeroDivisionError: division by zero
```

Python nie będzie również statycznie wykrywał innych problemów, takich jak niezdefiniowana metoda, zbyt duża lub zbyt mała liczba przekazanych argumentów, niepoprawny typ wartości zwrotnej i wiele innych, wydawałoby się oczywistych, problemów. Na szczęście społeczność opracowała różne narzędzia, które mogą pomóc w wychwytywaniu przynajmniej części tego rodzaju błędów. Przykładem takiego narzędzia jest linter `flake8` (*https://github.com/PyCQA/flake8*). Ponadto mamy narzędzia do sprawdzania typów, które współdziałają z wbudowanym modułem `typing` (więcej informacji na ten temat przedstawię w sposobie 124.).

Ostatecznie podczas tworzenia kodu zgodnego z duchem Pythona większość błędów napotkasz po uruchomieniu programu. Twórcy języka przywiązują większą wagę do elastyczności Pythona w trakcie działania programu niż do wykrywania błędów podczas kompilacji. Z tego powodu po uruchomieniu programu trzeba koniecznie sprawdzać poprawność przyjętych założeń (zobacz sposób 81.) oraz zweryfikować poprawność kodu źródłowego za pomocą testów zautomatyzowanych (zobacz sposób 109.).

Do zapamiętania

- Niemal wszystkie operacje sprawdzania pod kątem błędów Python odracza aż do chwili uruchomienia programu. To dotyczy również problemów, których istnienie wydaje się oczywiste już na etapie tworzenia programu.
- Opracowane przez społeczność projekty, takie jak lintery i narzędzia do statycznej analizy kodu źródłowego, mogą pomóc w wychwyceniu niektórych z najczęściej popełnianych błędów jeszcze przed uruchomieniem programu.

Programujesz w Pythonie?
Poznaj 125 sposobów na lepszy kod!

Python cieszy się dużym uznaniem. Jest wszechstronny i efektywny, a przy tym konsekwentnie rozwijany. Język ten ma również wiele trudniejszych do uchwycenia zalet. Aby uzyskać imponujące efekty wydajności kodu, jego wieloplatformowość i bezpieczeństwo, a przy tym w pełni korzystać z możliwości Pythona, potrzebujesz czegoś więcej niż tylko znajomość jego składni.

To trzecie, zaktualizowane i uzupełnione wydanie lubianego podręcznika programowania w Pythonie. Zawiera dodatkowe rozdziały dotyczące tak ważnych zagadnień jak algorytmy i struktury danych. Zawarty w książce materiał, w tym słowniki, ułatwi Ci tworzenie solidnego, wydajnego kodu. Znajdziesz tu jasne, zwięzłe i praktyczne rady przeznaczone dla programistów na różnym poziomie zaawansowania. Niezależnie od tego, czy tworzysz aplikacje internetowe, analizujesz dane, czy trenujesz modele sztucznej inteligencji, dzięki temu podręcznikowi zdobędziesz cenne umiejętności pracy z Pythonem!

W książce:

- 125 gotowych do użycia sposobów na lepszy kod
- uaktualnione dobre praktyki, uwzględniające nowości w Pythonie do wersji 3.13
- dodatkowe rozdziały poświęcone tworzeniu niezawodnych, wysoko wydajnych programów
- tworzenie modułów rozszerzeń w języku C
- współpraca z natywnymi bibliotekami współdzielonymi
- praktyczne przykłady kodu odzwierciedlające najlepsze praktyki

Brett Slatkin jest głównym inżynierem oprogramowania w firmie Google. Pracował nad projektem Google Surveys, nad protokołem PubSubHubbub, a wcześniej zajmował się pierwszym w Google produktem przetwarzania w chmurze — App Engine. Ukończył Uniwersytet Columbia w Nowym Jorku, programuje w Pythonie od blisko 20 lat.

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 978-83-289-2695-0

Cena: 139,00 zł

Pearson
Addison-Wesley