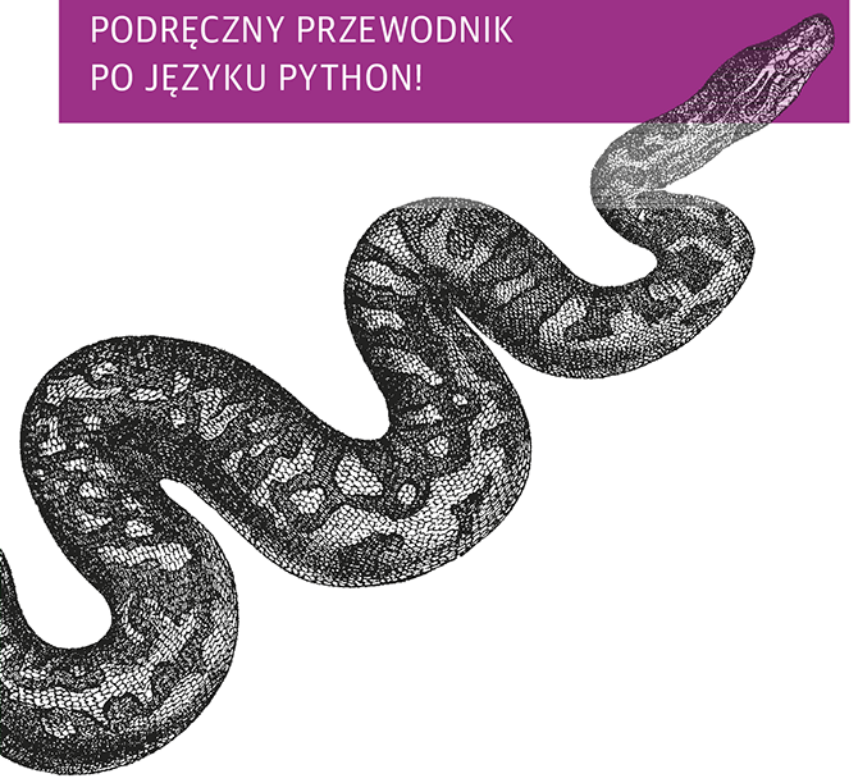
O'REILLY®

Wydanie V

Python

Leksykon kieszonkowy

PODRĘCZNY PRZEWODNIK
PO JĘZYKU PYTHON!



Helion

Mark Lutz

Tytuł oryginału: Python Pocket Reference, Fifth Edition

Tłumaczenie: Radosław Meryk

ISBN: 978-83-283-6035-8

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/pylk5v
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Instrukcje

W kolejnych podrozdziałach zostaną opisane wszystkie instrukcje występujące w Pythonie. W każdym podrozdziale zamieszczono format składni instrukcji, a za nim szczegółowe informacje na temat jej wykorzystania. W przypadku instrukcji złożonych każde wystąpienie ciągu *grupa* wewnątrz formatu instrukcji oznacza jedną lub więcej innych instrukcji. Instrukcje te mogą tworzyć blok pod wierszem nagłówka. Dla grupy trzeba zastosować wcięcie pod nagłówkiem, jeśli zawiera inną instrukcję złożoną (`if`, `while` itp.). W przeciwnym razie może się znaleźć w tym samym wierszu co nagłówek instrukcji. Obie poniższe konstrukcje są prawidłowe:

```
if x < 42:
    print(x)
    while x: x = x - 1

if x < 42: print(x)
```

Poniżej zamieszczono szczegółowe informacje wspólne dla wersji Pythona 3.X i 2.X. Szczegóły obowiązujące jedynie w wersji 2.X zestawiono na końcu podrozdziału „Instrukcje w Pythonie 2.X".

Instrukcja przypisania

```
cel = wyrażenie
cel1 = cel2 = wyrażenie
cel1, cel2 = wyrażenie1, wyrażenie2
cel1 += wyrażenie

cel1, cel2, ... = obiekt_iterowalny-o-tej-samej-długości
(cel1, cel2, ...) = obiekt_iterowalny-o-tej-samej-długości
[cel1, cel2, ...] = obiekt_iterowalny-o-tej-samej-długości
cel1, *cel2, ... = obiekt_iterowalny-o-odpowiedniej-długości
```

W instrukcjach przypisania cele zawierają *referencje* do obiektów. Instrukcje przypisania powinny mieć jawny format zamieszczony powyżej, w którym:

- *wyrażenia* generują obiekty;
- *cele* mogą być prostymi nazwami (`X`), kwalifikowanymi atrybutami (`X.attr`) albo indeksami i wycinkami (`X[i]`, `X[i:j:k]`);
- *zmienne* wewnątrz celów nie są deklarowane wcześniej, ale przed użyciem ich w wyrażeniu trzeba nadać im wartość (patrz podrozdział „Wyrażenia atomowe i dynamiczne określanie typów").

Pierwszy format wymieniony powyżej to *proste* przypisanie. Drugi z formatów — przypisanie *wielocelowe* (ang. *multiple-target*) — służy do przypisania tego samego wyrażenia do każdego z celów. Trzeci format — przypisanie *krotek* — łączy w pary cele z wyrażeniami, od lewej do prawej. Czwarty format — *przypisanie z aktualizacją* — jest skrótem dla połączenia operacji dodawania z przypisaniem (patrz następny podrozdział).

Ostatnie cztery formaty to *przypisanie sekwencji*, które służy do przypisywania komponentów dowolnej sekwencji (bądź innego obiektu iterowalnego) do odpowiadających im celów — od lewej do prawej. Sekwencja lub obiekt iterowalny występujące po prawej stronie mogą być dowolnego typu, ale muszą być tej samej długości, chyba że wśród celów z lewej strony znajdzie się nazwa rozpoczynająca się gwiazdką (`*X`), tak jak w ostatnim formacie. Ten ostatni format jest znany jako *rozszerzone przypisanie sekwencji*. Instrukcja o takim formacie jest dostępna wyłącznie w Pythonie 3.X. Pozwala ona na pobranie dowolnej liczby elementów (patrz podrozdział „Rozszerzone instrukcje przypisania sekwencji (3.X)").

Przypisania mogą również występować *niejawnie* w innych kontekstach w Pythonie (np. zmienne pętli `for` oraz mechanizm przekazywania argumentów funkcji). Niektóre z formatów instrukcji przypisania można również stosować w innych miejscach (np. przypisania sekwencji w instrukcjach `for`).

Przypisanie z aktualizacją

W Pythonie dostępny jest zbiór dodatkowych formatów instrukcji przypisania. Zestawiono je w tabeli 13. Są to tzw. *przypisania z aktualizacją* — formaty te implikują wyrażenie dwuargumentowe razem z przypisaniem. Na przykład poniższe dwa formaty w przybliżeniu są sobie równoważne:

```
X = X + Y
X += Y
```

Tabela 13. Instrukcje przypisania z aktualizacją

`X += Y`	`X &= Y`	`X -= Y`	`X |= Y`
`X *= Y`	`X ^= Y`	`X /= Y`	`X >>= Y`
`X %= Y`	`X <<= Y`	`X **= Y`	`X //= Y`

Jednak wartość celu `X` w instrukcji drugiego formatu może być wyznaczona *tylko raz*. Dodatkowo format ten pozwala na zastosowanie działań *w miejscu* dla typów mutowalnych (np. wyrażenie `list1 += list2` automatycznie wywołuje metodę `list1.extend(list2)` zamiast wolniejszej operacji konkatenacji implikowanej przez operator +). W klasach przypisania w miejscu mogą być przeciążane za pomocą nazw metod rozpoczynających się od `i` (np. `__iadd__()` dla operatora `+=`, `__add__()` dla operatora +). W Pythonie w wersji 2.2 wprowadzono nowy format: `X //= Y` (dzielenie całkowite).

Zwykłe instrukcje przypisania sekwencji

W Pythonie 2.X i 3.X dowolną sekwencję (bądź też inny obiekt iterowalny) złożoną z wartości można przypisać do dowolnej sekwencji nazw, pod warunkiem że ich długości są takie same. Podstawowa forma instrukcji przypisania sekwencji działa w większości kontekstów przypisania:

```
>>> a, b, c, d = [1, 2, 3, 4]
>>> a, d
(1, 4)

>>> for (a, b, c) in [[1, 2, 3], [4, 5, 6]]:
... print(a, b, c)
...
1 2 3
4 5 6
```

Rozszerzone instrukcje przypisania sekwencji (3.X)

W Pythonie 3.X (wyłącznie) instrukcja przypisania sekwencji została rozszerzona, by umożliwić obsługę przypisania sekwencji o dowolnej liczbie elementów — wystarczy poprzedzić gwiazdką jedną ze zmiennych celu przypisania. W takim przypadku długości sekwencji nie muszą być identyczne, natomiast nazwa poprzedzona gwiazdką pobiera niepasujące elementy do nowej listy:

```
>>> a, *b = [1, 2, 3, 4]
>>> a, b
(1, [2, 3, 4])
```

```
>>> a, *b, c = (1, 2, 3, 4)
>>> a, b, c
(1, [2, 3], 4)

>>> *a, b = 'spam'
>>> a, b
 (['s', 'p', 'a'], 'm')

>>> for (a, *b) in [[1, 2, 3], [4, 5, 6]]:
... print(a, b)
...
1 [2, 3]
4 [5, 6]
```

UWAGA

Uogólnienie gwiazdki w Pythonie 3.5 lub w wersjach późniejszych? W Pythonie 3.3 i wcześniejszych wersjach specjalne formy składni **x* i ***x* mogą występować w trzech miejscach: w *instrukcjach przypisania*, gdzie **x* pobiera niedopasowane elementy w instrukcjach przypisania sekwencji; w *nagłówkach funkcji*, w których te dwie formy służą do zbierania niepasujących argumentów pozycyjnych i argumentów w postaci słów kluczowych; oraz w *wywołaniach funkcji*, gdzie wymienione dwie formy rozpakowują obiekty iterowalne i słowniki do indywidualnych elementów (argumentów).

W Pythonie 3.4 deweloperzy planowali uogólnić składnię z gwiazdką w taki sposób, by można jej było używać także w *literałach opisujących struktury danych*, w których miałaby ona powodować rozpakowywanie kolekcji do postaci pojedynczych elementów, podobnie jak działa to obecnie w wywołaniach funkcji. Rozpakowująca składnia z gwiazdką będzie mogła być wykorzystywana dla *krotek*, *list*, *zbiorów*, *słowników* i *obiektów składanych*. Na przykład:

```
[x, *iter]          # rozpakowanie elementów obiektu iterowalnego iter: lista
(x, *iter), {x, *iter}   # to samo dla krotki, zbiór
{'x': 1, **dict}    # rozpakowanie elementów słownika: słowniki
[*iter for iter in x]    # rozpakowanie elementów obiektu iterowalnego:
                         obiekty składane
```

Są to dodatkowe miejsca zastosowania składni z gwiazdką oprócz instrukcji przypisania, nagłówków funkcji i wywołań funkcji. Dodatkowo być może zostaną zniesione niektóre obecne ograniczenia w zakresie korzystania z tej składni. Proponowana zmiana została przesunięta na wydanie po wersji 3.4, tuż przed ukazaniem się niniejszej książki, i pozostaje niepewna. W istocie dyskutowano o niej od 2008 roku. Zmiana ta nie będzie rozpatrywana do wydania Pythona 3.5 lub wersji późniejszej i być może nie pojawi się w ogóle. Szczegółowe informacje można znaleźć w dokumencie „What's New".

Instrukcja wyrażeniowa

```
wyrażenie
funkcja([wartość, nazwa=wartość, *nazwa, **nazwa...])
obiekt.metoda([wartość, nazwa=wartość, *nazwa, **nazwa...])
```

Dowolne wyrażenie może być instrukcją (np. występujące samodzielnie w wierszu). Z drugiej strony instrukcje nie mogą występować w żadnym innym kontekście wyrażenia (np. instrukcje przypisania nie zwracają wyników i nie mogą być zagnieżdżane).

Wyrażenia są powszechnie używane do wywoływania funkcji i metod, które nie zwracają wartości, oraz do wyświetlania w trybie interaktywnym. Instrukcje wyrażeniowe są także najpopularniejszym sposobem kodowania dla wyrażeń `yield` oraz wywołań wbudowanej funkcji `print()` z Pythona 3.X (choć w tej książce instrukcje te opisano osobno).

Składnia wywołania

W wywołaniach funkcji i metod aktualne argumenty wywołania są od siebie oddzielone przecinkami i zwykle są dopasowywane według pozycji do argumentów w nagłówkach `def` funkcji. W wywołaniach można opcjonalnie wymienić specyficzne nazwy argumentów, do których będą przekazywane wartości. W tym celu należy skorzystać ze składni argumentu kluczowego *`nazwa=wartość`*. Argumenty kluczowe są dopasowywane według nazwy zamiast pozycji.

Składnia dowolnych argumentów wywołania

W listach argumentów wywołania funkcji i metod można zastosować specjalną składnię w celu *rozpakowania* kolekcji do dowolnej liczby argumentów. Jeśli *`pargs`* i *`kargs`* to odpowiednio sekwencja (lub inny obiekt iterowalny) oraz słownik:

```
f(*pargs, **kargs)
```

to instrukcja o tym formacie wywoła funkcję `f` z argumentami *pozycyjnymi* z obiektu iterowalnego *`pargs`* oraz z *argumentami kluczowymi* ze słownika *`kargs`*. Na przykład:

```
>>> def f(a, b, c, d): print(a, b, c, d)
...
>>> f(*[1, 2], **dict(c=3, d=4))
1 2 3 4
```

Składnię tę dodano po to, by była symetryczna ze składnią argumentów nagłówka funkcji, na przykład `def f(*pargs, **kargs)`, która *pobiera*

niedopasowane argumenty. W wywołaniach elementy poprzedzone gwiazdkami są rozpakowywane do pojedynczych argumentów i mogą być łączone z innymi argumentami pozycyjnymi i kluczowymi zgodnie z regułami porządkowania (np. `g(1, 2, foo=3, bar=4, *pargs, **kargs)`).

W Pythonie 2.X podobny efekt można osiągnąć za pomocą wbudowanej funkcji `apply()`, która w Pythonie 3.X została usunięta:

```
apply(f, pargs, kargs)
```

Więcej informacji na temat składni wywołania można znaleźć w podrozdziale „Instrukcja def” (włącznie z tabelą 15.).

Instrukcja print

W Pythonie 3.X wyświetlanie tekstu do standardowego strumienia wyjściowego przyjęło postać wywołania wbudowanej funkcji. Zwykle jest ona kodowana jako instrukcja wyrażeniowa (w oddzielnym wierszu). Jej sygnatura wywołania jest następująca:

```
print([wartość [, wartość]*]
      [, sep=łańcuch-znaków] [, end=łańcuch-znaków]
      [, file=obiekt] [, flush=bool])
```

Każda *wartość* jest wyrażeniem generującym obiekt, dla którego ma być wyświetlony łańcuch znaków `str()`. To wywołanie konfiguruje się za pomocą trzech argumentów, które mogą być wyłącznie argumentami kluczowymi (jeśli argument będzie pominięty lub zostanie przekazana wartość `None`, przyjmie on wartość domyślną):

`sep`
: Łańcuch znaków, który ma być umieszczony pomiędzy wartościami (domyślnie spacja: `' '`).

`end`
: Łańcuch znaków do umieszczenia na końcu wyświetlanego tekstu (domyślnie znak nowego wiersza: `'\n'`).

`file`
: Obiekt postaci pliku, do którego jest zapisywany tekst (domyślnie standardowe wyjście: `sys.stdout`).

`flush`
: Wartość `true` lub `false` umożliwiająca włączenie lub wyłączenie wymuszonego opróżnienia strumienia wyjściowego (wprowadzone w Pythonie 3.3 — domyślnie `False`).

Aby wyłączyć separatory w postaci spacji oraz znaki wysuwu wiersza, można przekazać puste lub dostosowane do własnych potrzeb argumenty `sep` i `end`. W celu przekierowania wyjścia można przekazać nazwę pliku za pomocą argumentu `file` (patrz także podrozdział „Pliki”):

```
>>> print(2 ** 32, 'spam')
4294967296 spam

>>> print(2 ** 32, 'spam', sep='')
4294967296spam

>>> print(2 ** 32, 'spam', end=' '); print(1, 2, 3)
4294967296 spam 1 2 3

>>> print(2 ** 32, 'spam', sep='',
...                file=open('out', 'w'))
>>> open('out').read()
'4294967296spam\n'
```

Ponieważ domyślnie funkcja `print` po prostu wywołuje metodę `write()` obiektu, do którego w danym momencie odwołuje się urządzenie `sys.stdout`, to poniższa sekwencja jest równoważna wywołaniu `print(X)`:

```
import sys
sys.stdout.write(str(X) + '\n')
```

Aby przekierować polecenie wyświetlania tekstu do plików bądź obiektów klasy, należy albo przekazać dowolny obiekt z implementacją metody `write()` do argumentu kluczowego `file`, tak jak pokazano wcześniej, albo przypisać urządzenie `sys.stdout` do dowolnego takiego obiektu (patrz także podrozdział „Pliki”):

```
sys.stdout = open('log', 'a')  # dowolny obiekt z metodą write()
print('Ostrzeżenie: spam!')    # wywołanie jest kierowane do metody write() obiektu
```

Ponieważ do urządzenia wyjściowego `sys.stdout` można przypisać nową wartość, argument kluczowy `file` nie jest konieczny. Często jednak pozwala on na uniknięcie zarówno jawnych wywołań metody `write()`, jak i zapisywania i odtwarzania oryginalnej wartości `sys.stdout` wokół wywołania metody `print`, w przypadku gdy pierwotny strumień danych jest w dalszym ciągu wymagany. Więcej informacji na temat działania instrukcji `print()` w Pythonie 3.X można znaleźć w podrozdziale „Funkcje wbudowane”.

Instrukcja print w Pythonie 2.X

W Pythonie 2.X wyświetlanie jest instrukcją, a nie funkcją wbudowaną. Jest to instrukcja o następującej postaci:

```
print [wartość [, wartość]* [,]]
print >> plik [, wartość [, wartość]* [,]]
```

Instrukcja `print` z Pythona 2.X wyświetla drukowalne reprezentacje wartości w strumieniu `stdout` (używając bieżącego ustawienia `sys.stdout`) oraz dodaje spacje pomiędzy wartościami. Dodanie przecinka na końcu powoduje wyłączenie znaku wysuwu wiersza, który jest standardowo dodawany na końcu listy. Jest to równoważne użyciu klauzuli `end=' '` w Pythonie 3.X:

```
>>> print 2 ** 32, 'spam'
4294967296 spam

>>> print 2 ** 32, 'spam',; print 1, 2, 3
4294967296 spam 1 2 3
```

Instrukcja `print` z Pythona 2.X pozwala także na podanie nazwy pliku wyjściowego (lub podobnego mu obiektu), który będzie pełnił funkcję celu dla wyświetlanego tekstu zamiast strumienia `sys.stdout`.

```
fileobj = open('log', 'a')
print >> fileobj, "Ostrzeżenie: spam!"
```

Jeśli obiekt pliku ma wartość `None`, wykorzystywany jest strumień `sys.stdout`. Składnia Pythona 2.X `>>` jest równoważna użyciu argumentu kluczowego `file=F` z Pythona 3.X. W instrukcji `print` Pythona 2.X nie istnieje odpowiednik argumentu `sep=S`, chociaż wiersze mogą być sformatowane i wyświetlone jako pojedynczy element.

W instrukcji `print` Pythona 2.X można używać nawiasów okrągłych. W przypadku wielu elementów tworzą one krotki. Aby skorzystać z funkcji wyświetlania Pythona 3.X w Pythonie 2.X, należy uruchomić poniższe instrukcje (w interaktywnej sesji lub na początku skryptu) — z tego mechanizmu można skorzystać zarówno w *Pythonie 2.X* (w celu zachowania zgodności w przód z wersją 3.X), jak i w *Pythonie 3.X* (w celu zachowania wstecznej zgodności z Pythonem 2.X):

```
from __future__ import print_function
```

Instrukcja if

```
if warunek:
    grupa
[elif warunek:
    grupa]*
[else:
    grupa]
```

Instrukcja `if` pozwala na wybór jednego lub kilku działań (bloków instrukcji) i uruchamia grupę instrukcji powiązaną z pierwszym warun-

kiem `if` lub `elif`, który jest prawdziwy, albo wykonuje grupę `else`, jeśli wszystkie warunki `if` (`elif`) mają wartość `false`. Człony `elif` i `else` są opcjonalne.

Instrukcja while

```
while warunek:
    grupa
[else:
    grupa]
```

Pętla `while` to instrukcja pętli ogólnego przeznaczenia, która wykonuje pierwszą grupę instrukcji, gdy warunek na początku instrukcji jest prawdziwy. Instrukcja uruchamia grupę `else`, w przypadku gdy nastąpi zakończenie działania pętli bez wykonania instrukcji `break`.

Instrukcja for

```
for cel in obiekt_iterowalny:
    grupa
[else:
    grupa]
```

Pętla `for` realizuje iterację po sekwencji (bądź innym obiekcie iterowalnym). Przypisuje elementy obiektu iterowalnego do zmiennej *`cel`* i w każdej iteracji wykonuje pierwszą grupę instrukcji. Instrukcja `for` uruchamia opcjonalną grupę `else`, w przypadku gdy nastąpi zakończenie działania pętli bez wykonania instrukcji `break`. Zmienną *`cel`* instrukcji `for` może być dowolny obiekt, który może się znaleźć po lewej stronie instrukcji przypisania (np. `for (x, y) in lista_krotek`).

Od Pythona w wersji 2.2 instrukcja `for` najpierw próbuje uzyskać obiekt *iteratora* `I` za pomocą metody `iter(obiekt_iterowalny)`, a następnie wielokrotnie wywołuje metodę `I.__next__()` obiektu — do czasu wystąpienia wyjątku `StopIteration` (w Pythonie 2.X metoda `I.__next__()` ma nazwę `I.next()`). Jeśli nie można uzyskać żadnego obiektu iteratora (np. nie zdefiniowano metody `__iter__`), to instrukcja `for` działa poprzez wielokrotne indeksowanie obiektu *`obiekt_iterowalny`* z użyciem coraz większych wartości przesunięcia, aż zostanie zgłoszony wyjątek `IndexError`.

Iteracja w Pythonie odbywa się w wielu kontekstach, włącznie z instrukcjami pętli `for`, obiektami składanymi i instrukcjami `map()`. Więcej informacji na temat mechanizmów używanych przez pętlę `for` oraz w pozostałych kontekstach iteracji można znaleźć w podrozdziale „Protokół iteracji”.

Instrukcja pass

```
pass
```

Instrukcja-wypełniacz, która nie wykonuje żadnych działań. Używa się jej wtedy, gdy jest to syntaktycznie konieczne (np. w odniesieniu do namiastek funkcji). W Pythonie 3.X podobny efekt można uzyskać za pomocą wielokropka (`...`).

Instrukcja break

```
break
```

Powoduje natychmiastowe zakończenie najbliższej instrukcji pętli `while` lub `for` z pominięciem powiązanych z nimi grup `else` (o ile takie istnieją). Wskazówka: w celu wyjścia z wielopoziomowych pętli można skorzystać z instrukcji `raise` i `try`.

Instrukcja continue

```
continue
```

Powoduje natychmiastowe przejście na początek najbliższej instrukcji pętli `while` lub `for` i wznowienie wykonywania grupy instrukcji w pętli.

Instrukcja del

```
del nazwa
del nazwa[i]
del nazwa[i:j:k]
del nazwa.atrybut
```

Instrukcja `del` usuwa nazwy, elementy, wycinki i atrybuty, a także powiązania. W pierwszej postaci *`nazwa`* jest dosłowną nazwą zmiennej. W ostatnich trzech formach instrukcji *`nazwa`* może być dowolnym wyrażeniem (z użyciem nawiasów, jeśli są potrzebne do określenia priorytetu). Na przykład: `del a.b()[1].c.d`.

Instrukcja ta służy przede wszystkim do struktur danych, a nie zarządzania pamięcią. Usuwa również referencje do obiektów wywoływanych wcześniej. Może spowodować ich usunięcie przez mechanizm odśmiecania, w przypadku gdy nie będzie do nich odwołania w innych miejscach. Proces odśmiecania zachodzi jednak automatycznie i nie musi być wymuszany za pomocą instrukcji `del`.

Skorowidz

F

G

H

I

K

L

Ł

M

N

O

P

S

Ś

T

U

W

Z


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

O'REILLY®

Najlepsze rozwiązania typowych problemów!

Język Python błyskawicznie zdobył uznanie programistów na całym świecie. Sprawdza się doskonale w pisaniu skryptów oraz narzędzi, w dużym projekcie także nie zawiedzie oczekiwań. Python korzysta z automatycznego zarządzania pamięcią oraz umożliwia obiektowe i funkcyjne podejście do tworzonego programu. Liczna społeczność skupiona wokół tego języka na bieżąco wymienia się wiedzą na temat jego praktycznych zastosowań. Dzięki temu uzyskasz odpowiedzi na trapiące Cię pytania.

Przyda Ci się także sprawdzone źródło informacji. Ten przewodnik należy do serii *Leksykon kieszonkowy* i charakteryzuje się niezwykle zwięzłym, przejrzystym układem najważniejszych treści oraz poręczną formą. Znajdziesz tu szczegółowe informacje na temat typów wbudowanych, wyjątków, programowania obiektowego oraz przetwarzania nazw i reguł zasięgu. To wydanie zostało zaktualizowane o mnóstwo nowości, takich jak wykorzystanie Python Launcher w systemie Windows czy formalne reguły dziedziczenia. To doskonałe źródło wiedzy na temat języka Python!

Mark Lutz
— znany i ceniony na całym świecie instruktor programowania w Pythonie.

Dzięki tej książce:

- poznasz podstawy Pythona
- zapoznasz się z zasadami programowania w tym języku
- poznasz typy wbudowane
- wykorzystasz standardowe moduły

ISBN 978-83-283-6035-8

Cena: 39,90 zł