

Słowo wstępne	11
Przedmowa	13
1. Wydajny kod Python	19
Podstawowy system komputerowy	19
Jednostki obliczeniowe	20
Jednostki pamięci	23
Warstwy komunikacji	26
Łączenie ze sobą podstawowych elementów	27
Porównanie wyidealizowanego przetwarzania z maszyną wirtualną języka Python	27
Dlaczego warto używać języka Python?	31
Jak zostać bardzo wydajnym programistą?	34
Sprawdzone praktyki	35
Wnioski dotyczące sprawdzonych praktyk korzystania z rozszerzenia Jupyter Notebook	37
Niech praca znów sprawia radość	38
2. Użycie profilowania do znajdowania wąskich gardeł	39
Efektywne profilowanie	40
Wprowadzenie do zbioru Julii	41
Obliczanie pełnego zbioru Julii	45
Proste metody pomiaru czasu — instrukcja print i dekorator	48
Prosty pomiar czasu za pomocą polecenia time systemu Unix	51
Użycie modułu cProfile	52
Użycie narzędzia snakeviz do wizualizacji danych wyjściowych modułu cProfile	57
Użycie narzędzia line_profiler do pomiarów dotyczących kolejnych wierszy kodu	57
Użycie narzędzia memory_profiler do diagnozowania wykorzystania pamięci	63
Introspekcja istniejącego procesu za pomocą narzędzia py-spy	68

Kod bajtowy od podszewki	69
Użycie modułu <code>dis</code> do sprawdzenia kodu bajtowego narzędzia CPython	69
Różne metody, różna złożoność	71
Testowanie jednostkowe podczas optymalizacji w celu zachowania poprawności	73
Dekorator <code>@profile</code> bez operacji	73
Strategie udanego profilowania kodu	76
Podsumowanie	77
3. Listy i krotki	79
Bardziej efektywne wyszukiwanie	82
Porównanie list i krotek	84
Listy jako tablice dynamiczne	85
Krotki w roli tablic statycznych	88
Podsumowanie	89
4. Słowniki i zbiory	91
Jak działają słowniki i zbiory?	94
Wstawianie i pobieranie	94
Usuwanie	97
Zmiana wielkości	98
Funkcje mieszania i entropia	99
Słowniki i przestrzenie nazw	102
Podsumowanie	105
5. Iteratory i generatory	107
Iteratory dla szeregów nieskończonych	111
Wartościowanie leniwe generatora	112
Podsumowanie	116
6. Obliczenia macierzowe i wektorowe	117
Wprowadzenie do problemu	118
Czy listy języka Python są wystarczająco dobre?	122
Problemy z przesadną alokacją	123
Fragmentacja pamięci	126
Narzędzie <code>perf</code>	128
Podejmowanie decyzji z wykorzystaniem danych wyjściowych narzędzia <code>perf</code>	131
Wprowadzenie do narzędzia <code>numpy</code>	132
Zastosowanie narzędzia <code>numpy</code> w przypadku problemu dotyczącego dyfuzji	135
Przydziały pamięci i operacje wewnętrzne	138
Optymalizacje selektywne: znajdowanie tego, co wymaga poprawienia	141
Moduł <code>numexpr</code> : przyspieszanie i upraszczanie operacji wewnętrznych	143
Przestroga: weryfikowanie „optymalizacji” (biblioteka <code>scipy</code>)	146

Wnioski z optymalizacji macierzy	148
Narzędzie Pandas	150
Model wewnętrzny narzędzia Pandas	150
Zastosowanie funkcji dla wielu wierszy danych	152
Budowanie struktur DataFrame i szeregów z wyników częściowych, a nie przez łączenie	159
Zadanie może zostać zrealizowane na więcej niż jeden sposób (i być może szybszy)	160
Rada dotycząca efektywnego projektowania z wykorzystaniem narzędzia Pandas	161
Podsumowanie	163
7. Kompilowanie do postaci kodu C	165
Jakie wzrosty szybkości są możliwe?	166
Porównanie kompilatorów JIT i AOT	168
Dlaczego informacje o typie ułatwiają przyspieszenie działania kodu?	168
Użycie kompilatora kodu C	169
Analiza przykładu zbioru Julii	170
Cython	170
Kompilowanie czystego kodu Python za pomocą narzędzia Cython	171
pyximport	173
Użycie adnotacji kompilatora Cython do analizowania bloku kodu	173
Dodawanie adnotacji typu	175
Cython i numpy	179
Przetwarzanie równoległe rozwiązania na jednym komputerze z wykorzystaniem interfejsu OpenMP	181
Numba	183
Użycie narzędzia Numba do kompilacji kodu NumPy dla narzędzia Pandas	185
PyPy	186
Różnice związane z czyszczeniem pamięci	187
Uruchamianie interpretera PyPy i instalowanie modułów	188
Zestawienie wzrostów szybkości	189
Kiedy stosować poszczególne technologie?	190
Inne przyszełe projekty	192
Procesory graficzne (GPU)	193
Grafy dynamiczne: PyTorch	193
Podstawowe profilowanie procesora graficznego	196
Elementy wydajności procesorów graficznych	197
Kiedy stosować procesory graficzne?	199

Interfejsy funkcji zewnętrznych	201
ctypes	202
cffi	204
f2py	206
Moduł narzędzia CPython	209
Podsumowanie	212
8. Asynchroniczne operacje wejścia-wyjścia	215
Wprowadzenie do programowania asynchronicznego	217
Jak działają funkcja async i instrukcja await?	219
Przeszukiwacz szeregowy	220
gevent	222
tornado	226
aiohttp	229
Wspólne obciążenie procesora i urządzeń wejścia-wyjścia	232
Proces szeregowy	233
Przetwarzanie wsadowe wyników	234
Pełna asynchronizacja	237
Podsumowanie	240
9. Moduł multiprocessing	243
Moduł multiprocessing	246
Przybliżenie liczby pi przy użyciu metody Monte Carlo	248
Przybliżanie liczby pi za pomocą procesów i wątków	249
Zastosowanie obiektów języka Python	250
Zastępowanie modułu multiprocessing biblioteką Joblib	256
Liczby losowe w systemach przetwarzania równoległego	260
Zastosowanie narzędzia numpy	261
Znajdowanie liczb pierwszych	264
Kolejki zadań roboczych	270
Weryfikowanie liczb pierwszych za pomocą komunikacji międzyprocesowej	274
Rozwiązanie z przetwarzaniem szeregowym	279
Rozwiązanie z prostym obiektem Pool	280
Rozwiązanie z bardzo prostym obiektem Pool dla mniejszych liczb	281
Użycie obiektu Manager.Value jako flagi	282
Użycie systemu Redis jako flagi	284
Użycie obiektu RawValue jako flagi	286
Użycie modułu mmap jako flagi	287
Użycie modułu mmap do odtworzenia flagi	288
Współużytkowanie danych narzędzia numpy za pomocą modułu multiprocessing	290

Synchronizowanie dostępu do zmiennych i plików	297
Blokowanie plików	297
Blokowanie obiektu Value	300
Podsumowanie	303
10. Klastry i kolejki zadań	305
Zalety klastrowania	306
Wady klastrowania	307
Strata o wartości 462 milionów dolarów	
na giełdzie Wall Street z powodu kiepskiej strategii aktualizacji klastra	309
24-godzinny przestój usługi Skype w skali globalnej	309
Typowe projekty klastrowe	310
Metoda rozpoczęcia tworzenia rozwiązania klastrowego	311
Sposoby na uniknięcie kłopotów podczas korzystania z klastrów	312
Dwa rozwiązania klastrowe	313
Użycie modułu IPython Parallel do obsługi badań	314
Operacje równoległe narzędzia Pandas z wykorzystaniem biblioteki Dask	316
Użycie systemu NSQ dla niezawodnych klastrów produkcyjnych	321
Kolejki	321
Publikator/subskrybent	322
Rozproszone obliczenia liczb pierwszych	324
Inne warte uwagi narzędzia klastrowania	328
Docker	329
Wydajność Dockera	329
Zalety Dockera	332
Podsumowanie	334
11. Mniejsze wykorzystanie pamięci RAM	335
Obiekty typów podstawowych są kosztowne	336
Moduł array zużywa mniej pamięci	
do przechowywania wielu obiektów typu podstawowego	337
Mniejsze zużycie pamięci RAM w bibliotece NumPy dzięki narzędziu NumExpr	340
Analiza wykorzystania pamięci RAM w kolekcji	343
Bajty i obiekty Unicode	345
Efektywne przechowywanie zbiorów tekstowych w pamięci RAM	346
Zastosowanie metod dla 11 milionów tokenów	347
Modelowanie większej ilości tekstu	
za pomocą narzędzia FeatureHasher biblioteki scikit-learn	355
Wprowadzenie do narzędzi DictVectorizer i FeatureHasher	356
Porównanie narzędzi DictVectorizer i FeatureHasher	
w wypadku rzeczywistego problemu	358

Macierze rzadkie biblioteki SciPy	360
Wskazówki dotyczące mniejszego wykorzystania pamięci RAM	363
Probabilistyczne struktury danych	363
Obliczenia o bardzo dużym stopniu przybliżenia	
z wykorzystaniem jednobajtowego licznika Morrisa	365
Wartości k-minimum	367
Filtry Blooma	371
Licznik LogLog	376
Praktyczny przykład	380

12. Rady specjalistów z branży 385

Usprawnianie potoków inżynierii cech za pomocą biblioteki Feature-engine	385
Inżynieria cech w przypadku uczenia maszynowego	386
Trudne zadanie wdrażania potoków inżynierii cech	386
Wykorzystanie możliwości bibliotek open source języka Python	387
Biblioteka Feature-engine usprawnia budowanie	
i wdrażanie potoków inżynierii cech	388
Ułatwienie adaptacji nowego pakietu open source	389
Projektowanie, utrzymywanie i zachęcanie	
do uczestnictwa w rozwoju bibliotek open source	390
Bardzo wydajne zespoły danologów	391
Ile to potrwa?	392
Poznanie i planowanie	392
Zarządzanie oczekiwaniami i dostarczeniem produktu	393
Numba	395
Prosty przykład	395
Najlepsze praktyki i zalecenia	397
Uzyskiwanie pomocy	400
Optymalizowanie a myślenie	401
Narzędzie Social Media Analytics (SoMA) firmy Adaptive Lab (2014)	403
Język Python w firmie Adaptive Lab	404
Projekt narzędzia SoMA	404
Zastosowana metodologia projektowa	405
Serwisowanie systemu SoMA	405
Rada dla inżynierów z branży	406
Technika głębokiego uczenia prezentowana przez firmę RadimRehurek.com (2014)	406
Strzał w dziesiątkę	407
Rady dotyczące optymalizacji	409
Podsumowanie	411

Uczenie maszynowe o dużej skali gotowe do zastosowań produkcyjnych w firmie Lyst.com (2014)	412
Projekt klastra	412
Ewolucja kodu w szybko rozwijającej się nowej firmie	412
Budowanie mechanizmu rekomendacji	413
Raportowanie i monitorowanie	413
Rada	414
Analiza serwisu społecznościowego o dużej skali w firmie Smesh (2014)	414
Rola języka Python w firmie Smesh	414
Platforma	415
Dopasowywanie łańcuchów w czasie rzeczywistym z dużą wydajnością	415
Raportowanie, monitorowanie, debugowanie i wdrażanie	417
Interpreter PyPy zapewniający powodzenie systemów przetwarzania danych i systemów internetowych (2014)	418
Wymagania wstępne	419
Baza danych	419
Aplikacja internetowa	420
Mechanizm OCR i tłumaczenie	420
Dystrybucja zadań i procesy robocze	421
Podsumowanie	421
Kolejki zadań w serwisie internetowym Lanyrd.com (2014)	421
Rola języka Python w serwisie Lanyrd	422
Zapewnianie odpowiedniej wydajności kolejki zadań	423
Raportowanie, monitorowanie, debugowanie i wdrażanie	423
Rada dla programistów z branży	423