

Spis treści

Wprowadzenie 11

Podziękowania 15

O autorze 17

Rozdział 1. Programowanie zgodne z duchem Pythona 19

Sposób 1. Ustalenie używanej wersji Pythona 19

Sposób 2. Stosuj styl PEP 8 21

Sposób 3. Różnice między typami bytes i str 23

Sposób 4. Wybieraj interpolowane ciągi tekstowe f zamiast ciągów tekstowych formatowania w stylu C i funkcji str.format() 28

Sposób 5. Decyduj się na funkcje pomocnicze zamiast na skomplikowane wyrażenia 38

Sposób 6. Zamiast indeksowania wybieraj rozpakowanie wielu operacji przypisania 41

Sposób 7. Preferuj użycie funkcji enumerate() zamiast range() 44

Sposób 8. Używaj funkcji zip() do równoczesnego przetwarzania iteratorów 46

Sposób 9. Unikaj bloków else po pętlach for i while 48

Sposób 10. Unikaj powtórzeń w wyrażeniach przypisania 50

Rozdział 2. Lista i słownik 57

Sposób 11. Umiejętnie podziel sekwencje 57

Sposób 12. Unikaj użycia indeksów początek, koniec i wartości kroku w pojedynczej operacji podziału 60

Sposób 13. Wybieraj rozpakowanie typu catch-all zamiast tworzenia wycinków 62

Sposób 14. Używaj parametru key podczas sortowania według skomplikowanych kryteriów 66

Sposób 15. Zachowaj ostrożność, gdy polegasz na kolejności wstawiania elementów do obiektu typu dict 71

Sposób 16. Podczas obsługi brakujących kluczy słownika wybieraj funkcję `get()` zamiast operatora `in` i wyjątku `KeyError` 78

Sposób 17. Podczas obsługi brakujących elementów w wewnętrznym stanie wybieraj typ `defaultdict` zamiast metody `setdefault()` 82

Sposób 18. Wykorzystaj metodę `__missing__()` do tworzenia wartości domyślnych w zależności od klucza 84

Rozdział 3. Funkcje 89

Sposób 19. Gdy funkcja zwraca wiele wartości, nie rozpakowuj więcej niż trzech zmiennych 89

Sposób 20. Preferuj wyjątki zamiast zwrotu wartości `None` 92

Sposób 21. Zobacz, jak domknięcia współdziałają z zakresem zmiennej 95

Sposób 22. Zmniejszenie wizualnego zagmatwania za pomocą zmiennej liczby argumentów pozycyjnych 98

Sposób 23. Zdefiniowanie zachowania opcjonalnego za pomocą argumentów w postaci słów kluczowych 101

Sposób 24. Użycie `None` i docstring w celu dynamicznego określenia argumentów domyślnych 105

Sposób 25. Wymuszaj czytelność kodu, stosując jedynie argumenty w postaci słów kluczowych 108

Sposób 26. Dekoratory funkcji definiuj za pomocą `functools.wraps` 112

Rozdział 4. Konstrukcje składane i generatory 117

Sposób 27. Używaj list składanych zamiast funkcji `map()` i `filter()` 117

Sposób 28. Unikaj więcej niż dwóch wyrażeń na liście składanej 119

Sposób 29. Stosuj wyrażenia przypisania, aby unikać powielania zadań w konstrukcjach składanych 121

Sposób 30. Rozważ użycie generatorów, zamiast zwracać listy 124

Sposób 31. Podczas iteracji przez argumenty zachowuj postawę defensywną 126

Sposób 32. Rozważ użycie generatora wyrażeń dla dużych list składanych 131

Sposób 33. Twórz wiele generatorów za pomocą wyrażenia `yield from` 132

Sposób 34. Unikaj wstrzykiwania danych do generatorów za pomocą metody `send()` 135

Sposób 35. Unikaj w generatorach przejścia między stanami za pomocą metody `throw()` 140

Sposób 36. Rozważ stosowanie modułu `itertools` w pracy z iteratorami i generatorami 144

Rozdział 5. Klasy i interfejsy 151

Sposób 37. Twórz klasy, zamiast zagnieżdżać wiele poziomów typów wbudowanych 151

Sposób 38. Dla prostych interfejsów akceptuj funkcje zamiast klas 157

Sposób 39. Użycie polimorfizmu `@classmethod` w celu ogólnego tworzenia obiektów 160

Sposób 40. Inicjalizacja klasy nadrzędnej za pomocą wywołania `super()` 165

Sposób 41. Rozważ łączenie funkcjonalności za pomocą klas domieszek 169

Sposób 42. Preferuj atrybuty publiczne zamiast prywatnych 173

Sposób 43. Stosuj dziedziczenie po `collections.abc` w kontenerach typów niestandardowych 178

Rozdział 6. Metaklasy i atrybuty 183

Sposób 44. Używaj zwykłych atrybutów zamiast metod typu `getter` i `setter` 183

Sposób 45. Rozważ użycie `@property` zamiast refaktoryzacji atrybutów 187

Sposób 46. Stosuj deskryptory, aby wielokrotnie wykorzystywać metody udekorowane przez `@property` 191

Sposób 47. Używaj metod `__getattr__()`, `__getattribute__()` i `__setattr__()` dla opóźnionych atrybutów 196

Sposób 48. Sprawdzaj podklasy za pomocą `__init_subclass__` 201

Sposób 49. Rejestruj istniejące klasy za pomocą `__init_subclass__()` 208

Sposób 50. Adnotacje atrybutów klas dodawaj za pomocą metody `__set_name__()` 212

Sposób 51. Dla złożonych rozszerzeń klas wybieraj dekoratory klas zamiast metaklas 216

Rozdział 7. Współbieżność i równoległość 223

Sposób 52. Używaj modułu `subprocess` do zarządzania procesami potomnymi 224

Sposób 53. Użycie wątków dla operacji blokujących wejście- wyjście, unikanie równoległości 228

Sposób 54. Używaj klasy Lock, aby unikać stanu wyścigu w wątkach 232

Sposób 55. Używaj klasy Queue do koordynacji pracy między wątkami 236

Sposób 56. Naucz się rozpoznawać, kiedy współbieżność jest niezbędna 244

Sposób 57. Unikaj tworzenia nowych egzemplarzy Thread na żądanie fan-out 248

Sposób 58. Pamiętaj, że stosowanie Queue do obsługi współbieżności wymaga refaktoringu 252

Sposób 59. Rozważ użycie klasy ThreadPoolExecutor, gdy wątki są potrzebne do zapewnienia współbieżności 258

Sposób 60. Zapewnij wysoką współbieżność operacji wejścia-wyjścia dzięki użyciu współprogramów 260

Sposób 61. Naucz się przekazywać do asyncio wątkowane operacje wejścia-wyjścia 264

Sposób 62. Połączenie wątków i współprogramów w celu ułatwienia konwersji na wersję stosującą asyncio 274

Sposób 63. Maksymalizuj responsywność przez unikanie blokującej pętli zdarzeń asyncio 280

Sposób 64. Rozważ użycie concurrent.futures(), aby otrzymać prawdziwą równoległość 283

Rozdział 8. Niezawodność i wydajność 289

Sposób 65. Wykorzystanie zalet wszystkich bloków w konstrukcji try-except-else-finally 289

Sposób 66. Rozważ użycie poleceń contextlib i with w celu uzyskania wielokrotnego użycia konstrukcji try-finally 294

Sposób 67. Podczas obsługi czasu lokalnego używaj modułu datetime zamiast time 297

Sposób 68. Niezawodne użycie pickle wraz z copyreg 301

Sposób 69. Gdy ważna jest precyzja, używaj modułu decimal 307

Sposób 70. Przed optymalizacją przeprowadzaj profilowanie 310

Sposób 71. Wybieraj typ deque podczas tworzenia kolejek typu producent - konsument 314

Sposób 72. Podczas wyszukiwania danych w sortowanych sekwencjach stosuj moduł bisect 321

Sposób 73. W kolejkach priorytetowych używaj modułu heapq 323

Sposób 74. Podczas kopiowania zerowych obiektów typu bytes używaj egzemplarzy memoryview i bytearray 331

Rozdział 9. Testowanie i debugowanie 337

Sposób 75. Używaj ciągów tekstowych repr do debugowania danych wyjściowych 338

Sposób 76. W podklasach klasy TestCase sprawdzaj powiązane ze sobą zachowanie 341

Sposób 77. Izoluj testy od siebie za pomocą metod setUp(), tearDown(), setUpModule() i tearDownModule() 348

Sposób 78. Podczas testowania kodu zawierającego skomplikowane zależności korzystaj z imitacji 350

Sposób 79. Hermetyzuj zależności, aby ułatwić tworzenie imitacji i testowanie 357

Sposób 80. Rozważ interaktywne usuwanie błędów za pomocą pdb 361

Sposób 81. Stosuj moduł tracemalloc, aby poznać sposób użycia pamięci i wykryć jej wycieki 365

Rozdział 10. Współpraca 369

Sposób 82. Kiedy szukać modułów opracowanych przez społeczność? 369

Sposób 83. Używaj środowisk wirtualnych dla odizolowanych i powtarzalnych zależności 370

Sposób 84. Dla każdej funkcji, klasy i modułu utwórz docstring 375

Sposób 85. Używaj pakietów do organizacji modułów i dostarczania stabilnych API 380

Sposób 86. Rozważ użycie kodu o zasięgu modułu w celu konfiguracji środowiska wdrożenia 385

Sposób 87. Zdefiniuj główny wyjątek Exception w celu odizolowania komponentu wywołującego od API 387

Sposób 88. Zobacz, jak przerwać krąg zależności 391

Sposób 89. Rozważ użycie modułu warnings podczas refaktoryzacji i migracji kodu 395

Sposób 90. Rozważ stosowanie analizy statycznej za pomocą modułu typing w celu usuwania błędów 401