

Spis treści

Wprowadzenie	11
Podziękowania	17
O autorze	19
Rozdział 1. Programowanie zgodne z duchem Pythona	21
Sposób 1. Ustalenie używanej wersji Pythona	21
Sposób 2. Stosuj styl PEP 8	23
Sposób 3. Nigdy nie oczekuj, że Python wykryje błędy podczas kompilacji	26
Sposób 4. Decyduj się na funkcje pomocnicze zamiast na skomplikowane wyrażenia	28
Sposób 5. Zamiast indeksowania wybieraj rozpakowanie wielu operacji przypisania	31
Sposób 6. Krotkę jednoelementową zawsze umieszczaj w nawiasie	35
Sposób 7. W prostej logice osadzonej używaj wyrażen warunkowych	38
Sposób 8. Unikaj powtórzeń w wyrażeniach przypisania	42
Sposób 9. Stosuj polecenie match w celu destrukuryzacji kontroli przepływu, ale unikaj go, gdy wystarczające będzie polecenie if	48
Rozdział 2. Ciągi tekstowe i wycinki	56
Sposób 10. Różnice między typami bytes i str	56
Sposób 11. Wybieraj interpolowane ciągi tekstowe f zamiast ciągów tekstowych formatowania w stylu C i funkcji str.format()	61
Sposób 12. Różnice między wywołaniami repr i str podczas wyświetlania obiektów	72
Sposób 13. Wybieraj jawną konkatencję ciągu tekstowego zamiast niejawnej, zwłaszcza w przypadku listy	76
Sposób 14. Umiejętnie podziel sekwencje	80
Sposób 15. Unikaj użycia indeksów początek, koniec i wartości kroku w pojedynczej operacji podziału	82
Sposób 16. Wybieraj rozpakowanie typu catch-all zamiast tworzenia wycinków	85

Rozdział 3. Pętle i iteratory	89
Sposób 17. Preferuj użycie funkcji <code>enumerate()</code> zamiast <code>range()</code>	89
Sposób 18. Używaj funkcji <code>zip()</code> do równoczesnego przetwarzania iteratorów	91
Sposób 19. Unikaj bloków <code>else</code> po pętlach <code>for</code> i <code>while</code>	94
Sposób 20. Nigdy nie używaj zmiennych pętli <code>for</code> po zakończeniu jej działania	96
Sposób 21. Podczas iteracji przez argumenty zachowuj postawę defensywną	98
Sposób 22. Nigdy nie modyfikuj kontenerów podczas iteracji przez nie, lecz skorzystaj z kopii lub pamięci podręcznej	103
Sposób 23. Przekazuj iteratory do wywołań <code>any()</code> i <code>all()</code> w celu skrócenia logiki	108
Sposób 24. Rozważ stosowanie modułu <code>itertools</code> w pracy z iteratorami i generatorami	112
Rozdział 4. Słowniki	119
Sposób 25. Zachowaj ostrożność, gdy polegasz na kolejności wstawiania elementów do obiektu typu <code>dict</code>	119
Sposób 26. Podczas obsługi brakujących kluczy słownika wybieraj funkcję <code>get()</code> zamiast operatora <code>in</code> i wyjątku <code>KeyError</code>	126
Sposób 27. Podczas obsługi brakujących elementów w wewnętrznym stanie wybieraj typ <code>defaultdict</code> zamiast metody <code>setdefault()</code>	131
Sposób 28. Wykorzystaj metodę <code>__missing__()</code> do tworzenia wartości domyślnych w zależności od klucza	133
Sposób 29. Twórz klasy, zamiast zagnieżdżać wiele poziomów słowników, <code>list</code> i <code>krotek</code>	136
Rozdział 5. Funkcje	142
Sposób 30. Ustal, kiedy argumenty funkcji mogą być zmienione	142
Sposób 31. Zwróć obiekt, gdy funkcja ma rozpakować więcej niż trzy zmienne	145
Sposób 32. Preferuj wyjątki zamiast zwrotu wartości <code>None</code>	148
Sposób 33. Zobacz, jak domknięcia współdziałają z zakresem zmiennej	151
Sposób 34. Zmniejszenie wizualnego zagmatwania za pomocą zmiennej liczby argumentów pozycyjnych	155
Sposób 35. Zdefiniowanie zachowania opcjonalnego za pomocą argumentów w postaci słów kluczowych	158
Sposób 36. Użycie <code>None</code> i <code>docstring</code> w celu dynamicznego określenia argumentów domyślnych	162
Sposób 37. Wymuszaj czytelność kodu za pomocą argumentów w postaci słów kluczowych i argumentów jedynie pozycyjnych	165
Sposób 38. Dekoratory funkcji definiuj za pomocą <code>functools.wraps</code>	170
Sposób 39. Podczas łączenia funkcji preferuj <code>functools.partial</code> zamiast wyrażeń <code>lambda</code>	173

Rozdział 6. Konstrukcje składane i generatory	176
Sposób 40. Używaj list składanych zamiast funkcji <code>map()</code> i <code>filter()</code>	176
Sposób 41. Unikaj więcej niż dwóch wyrażeń na liście składanej	178
Sposób 42. Stosuj wyrażenia przypisania, aby unikać powielania zadań w konstrukcjach składanych	180
Sposób 43. Rozważ użycie generatorów, zamiast zwracać listy	183
Sposób 44. Rozważ użycie generatora wyrażeń dla dużych list składanych	186
Sposób 45. Twórz wiele generatorów za pomocą wyrażenia <code>yield from</code>	188
Sposób 46. Iteratory przekazuj generatorom jako argumenty zamiast za pomocą metody <code>send()</code>	190
Sposób 47. Przejścia między stanami obsługuj za pomocą klasy, zamiast używać metody <code>throw()</code>	195
Rozdział 7. Klasy i interfejsy	200
Sposób 48. Dla prostych interfejsów akceptuj funkcje zamiast klas	200
Sposób 49. Wybieraj zorientowany obiektowo polimorfizm zamiast funkcji z wywołaniami <code>isinstance()</code>	204
Sposób 50. W stylu programowania funkcyjnego używaj <code>functools singledispatch</code> zamiast polimorfizmu zorientowanego obiektowo	208
Sposób 51. Używaj typu <code>dataclasses</code> zamiast lekkich klas	215
Sposób 52. Użycie polimorfizmu <code>@classmethod</code> w celu ogólnego tworzenia obiektów	227
Sposób 53. Inicjalizacja klasy nadrzędnej za pomocą wywołania <code>super()</code>	231
Sposób 54. Rozważ łączenie funkcjonalności za pomocą klas domieszek	235
Sposób 55. Preferuj atrybuty publiczne zamiast prywatnych	240
Sposób 56. Wybieraj moduł <code>dataclasses</code> , zamiast tworzyć niemodyfikowalne obiekty	245
Sposób 57. Stosuj dziedziczenie po <code>collections.abc</code> w kontenerach typów niestandardowych	253
Rozdział 8. Metaklasy i atrybuty	258
Sposób 58. Używaj zwykłych atrybutów zamiast metod typu <code>getter</code> i <code>setter</code>	258
Sposób 59. Rozważ użycie <code>@property</code> zamiast refaktoryzacji atrybutów	262
Sposób 60. Stosuj deskryptory, aby wielokrotnie wykorzystywać metody udekorowane przez <code>@property</code>	266
Sposób 61. Używaj metod <code>__getattr__()</code> , <code>__getattribute__()</code> i <code>__setattr__()</code> dla opóźnionych atrybutów	271
Sposób 62. Sprawdzaj podklasy za pomocą <code>__init_subclass__</code>	277
Sposób 63. Rejestruj istniejące klasy za pomocą <code>__init_subclass__()</code>	283
Sposób 64. Adnotacje atrybutów klas dodawaj za pomocą metody <code>__set_name__()</code> ...	288
Sposób 65. Przemyśl kolejność definicji klasy, aby zdefiniować powiązania między atrybutami	292
Sposób 66. Dla złożonych rozszerzeń klas wybieraj dekoratory klas zamiast metaklas	298

Rozdział 9. Współbieżność i równoległość	304
Sposób 67. Używaj modułu subprocess do zarządzania procesami potomnymi	305
Sposób 68. Użycie wątków dla operacji blokujących wejście-wyjście, unikanie równoległości	309
Sposób 69. Używaj klasy Lock, aby unikać stanu wyścigu w wątkach	314
Sposób 70. Używaj klasy Queue do koordynacji pracy między wątkami	317
Sposób 71. Naucz się rozpoznawać, kiedy współbieżność jest niezbędna	327
Sposób 72. Unikaj tworzenia nowych egzemplarzy Thread na żądanie fan-out	331
Sposób 73. Pamiętaj, że stosowanie Queue do obsługi współbieżności wymaga refaktoringu	335
Sposób 74. Rozważ użycie klasy ThreadPoolExecutor, gdy wątki są potrzebne do zapewnienia współbieżności	341
Sposób 75. Zapewnij wysoką współbieżność operacji wejścia-wyjścia dzięki użyciu współprogramów	344
Sposób 76. Naucz się przekazywać do asyncio wątkowane operacje wejścia-wyjścia	348
Sposób 77. Połączenie wątków i współprogramów w celu ułatwienia konwersji na wersję stosującą asyncio	359
Sposób 78. Maksymalizuj responsywność przez unikanie blokującej pętli zdarzeń asyncio	365
Sposób 79. Rozważ użycie concurrent.futures(), aby otrzymać prawdziwą równoległość	369
Rozdział 10. Niezawodność	373
Sposób 80. Wykorzystaj zalety wszystkich bloków w konstrukcji try-except-else-finally	373
Sposób 81. Używaj polecenia assert dla wewnętrznych założeń i raise w przypadku niespełnionych oczekiwań	378
Sposób 82. Rozważ użycie poleceń contextlib i with w celu uzyskania wielokrotnego użycia konstrukcji try-finally	381
Sposób 83. Zawsze staraj się maksymalnie skrócić blok try	385
Sposób 84. Uważaj na znikające zmienne wyjątku	387
Sposób 85. Uważaj podczas przechwytywania klasy Exception	388
Sposób 86. Poznaj różnicę między klasami Exception i BaseException	391
Sposób 87. Używaj modułu traceback w celu dostarczania dokładniejszych informacji o wyjątkach	396
Sposób 88. Rozważ jawne łączenie wyjątków, aby otrzymać przejrzyste stosy wywołań	400
Sposób 89. Zawsze przekazuj zasoby do generatorów i nakazuj komponentom wywołującym przeprowadzanie operacji porządkowych	407
Sposób 90. Nigdy nie przypisuj wartości False zmiennej <code>__debug__</code>	412
Sposób 91. Unikaj wywołań <code>exec()</code> i <code>eval()</code> , o ile nie tworzysz narzędzia programistycznego	414

Rozdział 11. Wydajność	417
Sposób 92. Przed optymalizacją przeprowadzaj profilowanie	418
Sposób 93. Wydajność działania kodu o znaczeniu krytycznym optymalizuj za pomocą testów wydajności modułu timeit	423
Sposób 94. Dostrzegaj, kiedy i jak zastąpić Pythona innym językiem programowania	427
Sposób 95. Rozważ użycie modułu ctypes w celu sprawnej integracji z bibliotekami natywnymi	431
Sposób 96. Rozważ użycie modułów rozszerzeń, aby zmaksymalizować wydajność działania i ergonomię	436
Sposób 97. Aby skrócić czas uruchamiania programu, korzystaj ze wstępnie skompilowanego kodu bajtowego i buforowania systemu plików	442
Sposób 98. Stosuj wczytywanie modułów z opóźnieniem i importowanie dynamiczne, aby skrócić czas uruchamiania programu	445
Sposób 99. Rozważ użycie typów memoryview i bytearray, gdy podczas pracy z typem bytes stosujesz tzw. kopiowanie zero	451
Rozdział 12. Algorytmy i struktury danych	458
Sposób 100. Używaj parametru key podczas sortowania według skomplikowanych kryteriów	458
Sposób 101. Poznaj różnice między metodami sort() i sorted()	464
Sposób 102. Podczas wyszukiwania danych w sortowanych sekwencjach stosuj moduł bisect	465
Sposób 103. Wybieraj typ deque podczas tworzenia kolejek typu producent – konsument	468
Sposób 104. Przekonaj się, jak używać heapq w kolejce priorytetowej	473
Sposób 105. Podczas obsługi czasu lokalnego używaj modułu datetime zamiast time	481
Sposób 106. Gdy ważna jest precyzja, używaj modułu decimal	485
Sposób 107. Niezawodne użycie pickle wraz z copyreg	488
Rozdział 13. Testowanie i debugowanie	495
Sposób 108. W podklasach klasy TestCase sprawdzaj powiązane ze sobą zachowanie	496
Sposób 109. Wybieraj testy integracyjne zamiast testów jednostkowych	502
Sposób 110. Izoluj testy od siebie za pomocą metod setUp(), tearDown(), setUpModule() i tearDownModule()	507
Sposób 111. Podczas testowania kodu zawierającego skomplikowane zależności korzystaj z imitacji	509
Sposób 112. Hermetyzuj zależności, aby ułatwić tworzenie imitacji i testowanie	517

Sposób 113. Używaj metody <code>assertAlmostEqual()</code> do kontrolowania precyzji w testach liczb zmiennoprzecinkowych	520
Sposób 114. Rozważ interaktywne usuwanie błędów za pomocą <code>pdb</code>	523
Sposób 115. Stosuj moduł <code>tracemalloc</code> , aby poznać sposób użycia pamięci i wykryć jej wycieki	527
Rozdział 14. Współpraca	531
Sposób 116. Kiedy szukać modułów opracowanych przez społeczność?	531
Sposób 117. Używaj środowisk wirtualnych dla odizolowanych i powtarzalnych zależności	532
Sposób 118. Dla każdej funkcji, klasy i modułu utwórz <code>docstring</code>	537
Sposób 119. Używaj pakietów do organizacji modułów i dostarczania stabilnych API	542
Sposób 120. Rozważ użycie kodu o zasięgu modułu w celu konfiguracji środowiska wdrożenia	547
Sposób 121. Zdefiniuj główny wyjątek <code>Exception</code> w celu odizolowania komponentu wywołującego od API	549
Sposób 122. Zobacz, jak przerwać krąg zależności	553
Sposób 123. Rozważ użycie modułu <code>warnings</code> podczas refaktoryzacji i migracji kodu	558
Sposób 124. Rozważ stosowanie analizy statycznej za pomocą modułu <code>typing</code> w celu usuwania błędów	565
Sposób 125. Podczas tworzenia paczek programów w Pythonie wybieraj projekty otwartoźródłowe zamiast modułów <code>zipimport</code> i <code>zipapp</code>	571