

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, str i unicode	23
Sposób 4. Decyduj się na funkcje pomocnicze zamiast na skomplikowane wyrażenia	26
Sposób 5. Umiejętnie podziel sekwencje	29
Sposób 6. Unikaj użycia indeksów początek, koniec i wartości kroku w pojedynczej operacji podziału	31
Sposób 7. Używaj list składanych zamiast funkcji map() i filter()	33
Sposób 8. Unikaj więcej niż dwóch wyrażen na liście składanej	35
Sposób 9. Rozważ użycie generatora wyrażen dla dużych list składanych	36
Sposób 10. Preferuj użycie funkcji enumerate() zamiast range()	38
Sposób 11. Użycie funkcji zip() do równoczesnego przetwarzania iteratorów ...	39
Sposób 12. Unikaj bloków else po pętlach for i while	41
Sposób 13. Wykorzystanie zalet wszystkich bloków w konstrukcji try-except-else-finally	44
Rozdział 2. Funkcje	47
Sposób 14. Preferuj wyjątki zamiast zwrotu wartości None	47
Sposób 15. Zobacz, jak domknięcia współdziałają z zakresem zmiennej	49
Sposób 16. Rozważ użycie generatorów, zamiast zwracać listy	54

Sposób 17. Podczas iteracji przez argumenty zachowuj postawę defensywną	56
Sposób 18. Zmniejszenie wizualnego zagmatwania za pomocą zmiennej liczby argumentów pozycyjnych	61
Sposób 19. Zdefiniowanie zachowania opcjonalnego za pomocą argumentów w postaci słów kluczowych	63
Sposób 20. Użycie None i docstring w celu dynamicznego określenia argumentów domyślnych	66
Sposób 21. Wymuszaj czytelność kodu, stosując jedynie argumenty w postaci słów kluczowych	69

Rozdział 3. Klasy i dziedziczenie 73

Sposób 22. Preferuj klasy pomocnicze zamiast słowników i krotek	73
Sposób 23. Dla prostych interfejsów akceptuj funkcje zamiast klas	78
Sposób 24. Użycie polimorfizmu @classmethod w celu ogólnego tworzenia obiektów	82
Sposób 25. Inicjalizacja klasy nadrzędnej za pomocą wywołania super()	87
Sposób 26. Wielokrotnego dziedziczenia używaj jedynie w klasach narzędziowych	91
Sposób 27. Preferuj atrybuty publiczne zamiast prywatnych	95
Sposób 28. Dziedziczenie po collections.abc w kontenerach typów niestandardowych	99

Rozdział 4. Metaklasy i atrybuty 105

Sposób 29. Używaj zwykłych atrybutów zamiast metod typu getter i setter ...	105
Sposób 30. Rozważ użycie @property zamiast refaktoryzacji atrybutów	109
Sposób 31. Stosuj deskryptory, aby wielokrotnie wykorzystywać metody udekorowane przez @property	113
Sposób 32. Używaj metod __getattr__(), __getattribute__() i __setattr__() dla opóźnionych atrybutów	117
Sposób 33. Sprawdzaj podklasy za pomocą metaklas	122
Sposób 34. Rejestruj istniejące klasy wraz z metaklasami	124
Sposób 35. Adnotacje atrybutów klas dodawaj za pomocą metaklas	128

Rozdział 5. Współbieżność i równoległość 131

Sposób 36. Używaj modułu subprocess do zarządzania procesami potomnymi	132
Sposób 37. Użycie wątków dla operacji blokujących wejście-wyjście, unikanie równoległości	136
Sposób 38. Używaj klasy Lock, aby unikać stanu wyścigu w wątkach	140
Sposób 39. Używaj klasy Queue do koordynacji pracy między wątkami	143

Sposób 40. Rozważ użycie współprogramów w celu jednoczesnego wykonywania wielu funkcji	150
Sposób 41. Rozważ użycie <code>concurrent.futures()</code> , aby otrzymać prawdziwą równoległość	158

Rozdział 6. Wbudowane moduły163

Sposób 42. Dekoratory funkcji definiuj za pomocą <code>functools.wraps</code>	163
Sposób 43. Rozważ użycie poleceń <code>contextlib</code> i <code>with</code> w celu uzyskania wielokrotnego użycia konstrukcji <code>try-finally</code>	166
Sposób 44. Niezawodne użycie <code>pickle</code> wraz z <code>copyreg</code>	169
Sposób 45. Podczas obsługi czasu lokalnego używaj modułu <code>datetime</code> zamiast <code>time</code>	174
Sposób 46. Używaj wbudowanych algorytmów i struktur danych	178
Sposób 47. Gdy ważna jest precyzja, używaj modułu <code>decimal</code>	183
Sposób 48. Kiedy szukać modułów opracowanych przez społeczność?	185

Rozdział 7. Współpraca187

Sposób 49. Dla każdej funkcji, klasy i modułu utwórz <code>docstring</code>	187
Sposób 50. Używaj pakietów do organizacji modułów i dostarczania stabilnych API	191
Sposób 51. Zdefiniuj główny wyjątek <code>Exception</code> w celu odizolowania komponentu wywołującego od API	196
Sposób 52. Zobacz, jak przerwać krąg zależności	199
Sposób 53. Używaj środowisk wirtualnych dla odizolowanych i powtarzalnych zależności	204

Rozdział 8. Produkcja211

Sposób 54. Rozważ użycie kodu o zasięgu modułu w celu konfiguracji środowiska wdrożenia	211
Sposób 55. Używaj ciągów tekstowych <code>repr</code> do debugowania danych wyjściowych	214
Sposób 56. Testuj wszystko za pomocą <code>unittest</code>	217
Sposób 57. Rozważ interaktywne usuwanie błędów za pomocą <code>pdb</code>	220
Sposób 58. Przed optymalizacją przeprowadzaj profilowanie	222
Sposób 59. Stosuj moduł <code>tracemalloc</code> , aby poznać sposób użycia pamięci i wykryć jej wycieki	226

Skorowidz 229