

# Spis treści

|                                                                |           |
|----------------------------------------------------------------|-----------|
| <b>O autorze</b>                                               | <b>9</b>  |
| <b>O recenzencie</b>                                           | <b>10</b> |
| <b>Przedmowa</b>                                               | <b>11</b> |
| <b>Rozdział 1. Wprowadzenie, formatowanie kodu i narzędzia</b> | <b>15</b> |
| <b>Wprowadzenie</b>                                            | <b>16</b> |
| Znaczenie terminu czysty kod                                   | 16        |
| Znaczenie posiadania czystego kodu                             | 17        |
| Kilka wyjątków                                                 | 18        |
| <b>Formatowanie kodu</b>                                       | <b>19</b> |
| Przestrzeganie przewodnika stylu kodowania w projekcie         | 20        |
| <b>Dokumentacja</b>                                            | <b>22</b> |
| Komentarze do kodu                                             | 23        |
| Docstringi                                                     | 24        |
| Adnotacje                                                      | 26        |
| Czy adnotacje zastępują docstringi?                            | 29        |
| <b>Narzędzia</b>                                               | <b>31</b> |
| Sprawdzanie spójności typów                                    | 32        |
| Ogólne sprawdzanie poprawności w kodzie                        | 34        |
| Formatowanie automatyczne                                      | 35        |
| Konfiguracja automatycznych kontroli                           | 38        |
| <b>Podsumowanie</b>                                            | <b>39</b> |
| <b>Materiały referencyjne</b>                                  | <b>40</b> |

|                                                          |            |
|----------------------------------------------------------|------------|
| <b>Rozdział 2. Kod pythoniczny</b>                       | <b>41</b>  |
| <b>Indeksy i wycinki</b>                                 | <b>42</b>  |
| Tworzenie własnych sekwencji                             | 43         |
| <b>Menedżery kontekstu</b>                               | <b>45</b>  |
| Implementacja menedżerów kontekstu                       | 48         |
| <b>Wyrażenia składane i wyrażenia przypisania</b>        | <b>51</b>  |
| <b>Właściwości, atrybuty i różne typy metod obiektów</b> | <b>53</b>  |
| Znaki podkreślenia w Pythonie                            | 53         |
| Właściwości                                              | 56         |
| Tworzenie klas o bardziej zwartej składni                | 58         |
| Obiekty iterowalne                                       | 61         |
| Obiekty kontenerowe                                      | 66         |
| Dynamiczne atrybuty obiektów                             | 67         |
| Obiekty wywoływalne                                      | 69         |
| Podsumowanie metod magicznych                            | 70         |
| <b>Haczki Pythona</b>                                    | <b>71</b>  |
| Mutowalne argumenty domyślne                             | 72         |
| Rozszerzanie typów wbudowanych                           | 73         |
| <b>Krótkie wprowadzenie do kodu asynchronicznego</b>     | <b>75</b>  |
| <b>Podsumowanie</b>                                      | <b>77</b>  |
| <b>Materiały referencyjne</b>                            | <b>77</b>  |
| <b>Rozdział 3. Ogólne cechy dobrego kodu</b>             | <b>79</b>  |
| <b>Projektowanie według kontraktu</b>                    | <b>80</b>  |
| Warunki wstępne                                          | 82         |
| Warunki końcowe                                          | 82         |
| Kontrakty pythoniczne                                    | 83         |
| Projektowanie według kontraktu — wnioski                 | 83         |
| <b>Programowanie defensywne</b>                          | <b>84</b>  |
| Obsługa błędów                                           | 84         |
| Używanie asercji w Pythonie                              | 92         |
| <b>Podział obowiązków</b>                                | <b>94</b>  |
| Spójność i sprzężenie                                    | 95         |
| <b>Akronimy</b>                                          | <b>96</b>  |
| DRY/OAOC                                                 | 96         |
| YAGNI                                                    | 98         |
| KIS                                                      | 99         |
| EAFP/LBYL                                                | 101        |
| <b>Dziedziczenie w Pythonie</b>                          | <b>102</b> |
| Kiedy zastosowanie dziedziczenia jest dobrą decyzją?     | 102        |
| Antywzorce dziedziczenia                                 | 103        |
| Wielokrotne dziedziczenie w Pythonie                     | 106        |
| <b>Argumenty funkcji i metod</b>                         | <b>109</b> |
| Jak działają argumenty funkcji w Pythonie?               | 109        |
| Liczba argumentów w funkcjach                            | 117        |

|                                                                             |            |
|-----------------------------------------------------------------------------|------------|
| <b>Uwagi końcowe dotyczące dobrych praktyk projektowania oprogramowania</b> | <b>120</b> |
| Ortogonalność w oprogramowaniu                                              | 120        |
| Strukturyzacja kodu                                                         | 122        |
| <b>Podsumowanie</b>                                                         | <b>123</b> |
| <b>Materiały referencyjne</b>                                               | <b>124</b> |
| <b>Rozdział 4. Zasady SOLID</b>                                             | <b>125</b> |
| <b>Zasada pojedynczej odpowiedzialności</b>                                 | <b>125</b> |
| Klasa mająca zbyt wiele obowiązków                                          | 126        |
| Podział obowiązków                                                          | 128        |
| <b>Zasada otwartej-zamkniętej</b>                                           | <b>129</b> |
| Przykład zagrożeń dla utrzymania kodu                                       | 130        |
| w przypadku nieprzestrzegania zasady OCP                                    | 130        |
| Refaktoryzacja systemu obsługi zdarzeń w celu uzyskania rozszerzalności     | 132        |
| Rozbudowa systemu zdarzeń                                                   | 134        |
| Końcowe przemyślenia na temat OCP                                           | 135        |
| <b>Zasada podstawiania Liskov</b>                                           | <b>135</b> |
| Wykrywanie problemów dotyczących zasady LSP za pomocą narzędzi              | 136        |
| Bardziej subtelne przypadki naruszeń zasady LSP                             | 139        |
| Uwagi na temat LSP                                                          | 141        |
| <b>Segregacja interfejsów</b>                                               | <b>142</b> |
| Interfejs, który dostarcza zbyt wiele                                       | 143        |
| Im mniejszy interfejs, tym lepiej                                           | 143        |
| Jak mały powinien być interfejs?                                            | 145        |
| <b>Odwracanie zależności</b>                                                | <b>145</b> |
| Przypadek sztywnych zależności                                              | 146        |
| Odwracanie zależności                                                       | 147        |
| Wstrzykiwanie zależności                                                    | 148        |
| <b>Podsumowanie</b>                                                         | <b>150</b> |
| <b>Bibliografia</b>                                                         | <b>151</b> |
| <b>Rozdział 5. Korzystanie z dekoratorów do usprawniania kodu</b>           | <b>152</b> |
| <b>Czym są dekoratory w Pythonie?</b>                                       | <b>153</b> |
| Dekoratory funkcji                                                          | 154        |
| Dekoratory klas                                                             | 155        |
| Inne rodzaje dekoratorów                                                    | 158        |
| <b>Bardziej zaawansowane dekoratory</b>                                     | <b>159</b> |
| Przekazywanie argumentów do dekoratorów                                     | 159        |
| Dekoratory z wartościami domyślnymi                                         | 163        |
| Dekoratory dla podprogramów                                                 | 165        |
| Rozszerzona składnia dekoratorów                                            | 167        |
| <b>Dobre zastosowania dla dekoratorów</b>                                   | <b>168</b> |
| Dostosowywanie sygnatur funkcji                                             | 169        |
| Walidacja parametrów                                                        | 170        |
| Śledzenie kodu                                                              | 170        |
| <b>Skuteczne dekoratory — unikanie typowych błędów</b>                      | <b>171</b> |
| Zachowywanie danych o oryginalnym opakowanym obiekcie                       | 171        |
| Obsługa skutków ubocznych w dekoratorach                                    | 173        |
| Tworzenie dekoratorów, które będą działać dla każdego rodzaju obiektów      | 177        |

|                                                                            |            |
|----------------------------------------------------------------------------|------------|
| <b>Dekoratory a czysty kod</b>                                             | <b>179</b> |
| Kompozycja zamiast dziedziczenia                                           | 180        |
| Zasada DRY z wykorzystaniem dekoratorów                                    | 182        |
| Dekoratory a podział odpowiedzialności                                     | 183        |
| Analiza dobrych dekoratorów                                                | 185        |
| <b>Podsumowanie</b>                                                        | <b>186</b> |
| <b>Bibliografia</b>                                                        | <b>187</b> |
| <b>Rozdział 6. Pełniejsze wykorzystywanie obiektów dzięki deskryptorom</b> | <b>188</b> |
| <b>Pierwsze spojrzenie na deskryptory</b>                                  | <b>189</b> |
| Oprządzanie związane z deskryptorami                                       | 189        |
| Opis metod protokołu deskryptora                                           | 192        |
| <b>Rodzaje deskryptorów</b>                                                | <b>198</b> |
| Deskryptory niezwiązane z danymi                                           | 199        |
| Deskryptory danych                                                         | 200        |
| <b>Deskryptory w praktyce</b>                                              | <b>202</b> |
| Zastosowanie deskryptorów                                                  | 203        |
| <b>Różne formy implementacji deskryptorów</b>                              | <b>207</b> |
| Problem współdzielonego stanu                                              | 207        |
| Dostęp do słownika obiektu                                                 | 208        |
| Korzystanie ze słabych referencji                                          | 208        |
| <b>Więcej uwag na temat deskryptorów</b>                                   | <b>209</b> |
| <b>Analiza deskryptorów</b>                                                | <b>213</b> |
| W jaki sposób Python wewnętrznie używa deskryptorów?                       | 214        |
| Implementacja deskryptorów w dekoratorach                                  | 219        |
| <b>Uwagi końcowe na temat deskryptorów</b>                                 | <b>220</b> |
| Interfejs deskryptorów                                                     | 220        |
| Obiektowy projekt deskryptorów                                             | 220        |
| Adnotacje typów dla deskryptorów                                           | 221        |
| <b>Podsumowanie</b>                                                        | <b>221</b> |
| <b>Bibliografia</b>                                                        | <b>222</b> |
| <b>Rozdział 7. Generatory, iteratory i programowanie asynchroniczne</b>    | <b>223</b> |
| <b>Wymagania techniczne</b>                                                | <b>224</b> |
| <b>Tworzenie generatorów</b>                                               | <b>224</b> |
| Pierwsze spojrzenie na generatory                                          | 224        |
| Wyrażenia generatorowe                                                     | 227        |
| <b>Idiomatyczne iteracje</b>                                               | <b>228</b> |
| Idiomy iteracji                                                            | 229        |
| Funkcja next()                                                             | 230        |
| Korzystanie z generatora                                                   | 231        |
| Itertools                                                                  | 232        |
| Upraszczanie kodu za pomocą iteratorów                                     | 233        |
| Wzorzec Iterator w Pythonie                                                | 235        |
| <b>Podprogramy</b>                                                         | <b>239</b> |
| Metody interfejsu generatora                                               | 239        |
| Bardziej zaawansowane podprogramy                                          | 244        |

|                                                              |            |
|--------------------------------------------------------------|------------|
| <b>Programowanie asynchroniczne</b>                          | <b>250</b> |
| Magiczne metody asynchroniczne                               | 252        |
| Iteracja asynchroniczna                                      | 254        |
| Generatory asynchroniczne                                    | 257        |
| <b>Podsumowanie</b>                                          | <b>258</b> |
| <b>Bibliografia</b>                                          | <b>258</b> |
| <b>Rozdział 8. Testy jednostkowe i refaktoryzacja</b>        | <b>260</b> |
| <b>Zasady projektowania a testy jednostkowe</b>              | <b>261</b> |
| Uwaga na temat innych form automatycznych testów             | 262        |
| Testy jednostkowe a zwinne wytwarzanie oprogramowania        | 263        |
| Testy jednostkowe a projektowanie oprogramowania             | 264        |
| Definiowanie granic testowania                               | 267        |
| <b>Narzędzia testowania</b>                                  | <b>268</b> |
| Frameworki i biblioteki do testów jednostkowych              | 268        |
| <b>Refaktoryzacja</b>                                        | <b>287</b> |
| Ewolucje kodu                                                | 287        |
| Kod produkcyjny nie jest jedynym, który ewoluuje             | 289        |
| <b>Więcej o testowaniu</b>                                   | <b>290</b> |
| Testowanie oparte na właściwościach                          | 291        |
| Testowanie mutacji                                           | 291        |
| Typowe motywy w testowaniu                                   | 293        |
| Krótkie wprowadzenie do techniki TDD                         | 295        |
| <b>Podsumowanie</b>                                          | <b>296</b> |
| <b>Bibliografia</b>                                          | <b>297</b> |
| <b>Rozdział 9. Typowe wzorce projektowe</b>                  | <b>298</b> |
| <b>Zagadnienia dotyczące wzorców projektowych w Pythonie</b> | <b>299</b> |
| <b>Wzorce projektowe w praktyce</b>                          | <b>300</b> |
| Wzorce kreacyjne                                             | 301        |
| Wzorce strukturalne                                          | 307        |
| Wzorce behawioralne                                          | 313        |
| <b>Pusty obiekt</b>                                          | <b>323</b> |
| <b>Końcowe przemyślenia dotyczące wzorców projektowych</b>   | <b>325</b> |
| Wpływ zastosowania wzorców na projekt                        | 325        |
| Wzorce projektowe jako teoria                                | 326        |
| Nazwy w modelach                                             | 327        |
| <b>Podsumowanie</b>                                          | <b>327</b> |
| <b>Bibliografia</b>                                          | <b>328</b> |
| <b>Rozdział 10. Czysta architektura</b>                      | <b>329</b> |
| <b>Od czystego kodu do czystej architektury</b>              | <b>330</b> |
| Podział odpowiedzialności                                    | 330        |
| Aplikacje monolityczne a mikroustugi                         | 332        |
| Abstrakcje                                                   | 333        |

