

## Spis treści

O autorach (11)

O recenzencie (12)

Przedmowa (13)

### Rozdział 1. Obecny status Pythona (19)

Gdzie jesteśmy i dokąd zmierzamy? (20)

Dlaczego i jak zmienia się Python (20)

Bądź na bieżąco ze zmianami języka - dokumenty PEP (21)

Popularność Pythona 3 w chwili pisanie tej książki (22)

Główne różnice pomiędzy Pythonem 3 a Pythonem 2 (23)

Korzyści płynące ze znajomości starej wersji Pythona (23)

Główne różnice składni i częste pułapki (24)

Popularne narzędzia i techniki używane w celu utrzymania kompatybilności (26)

Nie tylko CPython (30)

Dlaczego powinieneś się przejmować? (30)

Stackless Python (31)

Jython (31)

IronPython (32)

PyPy (33)

Nowoczesne podejścia do programowania w Pythonie (34)

Izolacja środowisk Pythona na poziomie aplikacji (34)

Zalety stosowania izolacji (36)

Popularne rozwiązania (37)

Które rozwiązanie wybrać? (41)

Izolacja środowisk Pythona na poziomie systemu operacyjnego (42)

Wirtualne środowiska robocze z wykorzystaniem narzędzia Vagrant (43)

Konteneryzacja czy wirtualizacja? (45)

Popularne narzędzia pracy ukierunkowane na produktywność (45)

Alternatywne powłoki Pythona - IPython, bpython, ptpython (46)

Interaktywne debuggery (48)

Przydatne materiały (49)

Podsumowanie (50)

### Rozdział 2. Najlepsze praktyki składniowe - poniżej poziomu klas (51)

Typy wbudowane Pythona (52)

Ciągi znaków i bajtów (52)

Kolekcje (56)

Zaawansowane elementy składni (67)

Iteratory (67)

Instrukcja yield (69)

Dekoratory (72)

Zarządcy kontekstu - instrukcja with (83)

Inne elementy składni, o których możesz jeszcze nie wiedzieć (87)

Konstrukcja for ... else ... (87)

Adnotacje funkcji (88)

Podsumowanie (89)

### Rozdział 3. Najlepsze praktyki składniowe - powyżej poziomu klas (91)

|                                                                                                      |  |
|------------------------------------------------------------------------------------------------------|--|
| Dziedziczenie po typach wbudowanych (92)                                                             |  |
| Uzyskiwanie dostępu do metod klas nadrzędnych (94)                                                   |  |
| Klasy w starym stylu oraz funkcja <code>super()</code> w Pythonie 2 (96)                             |  |
| Porządek rozpatrywania metod w Pythonie (97)                                                         |  |
| Pułapki związane z funkcją <code>super()</code> (101)                                                |  |
| Najlepsze praktyki (104)                                                                             |  |
| Zaawansowane wzorce dostępu do atrybutów (104)                                                       |  |
| Deskryptory (105)                                                                                    |  |
| Właściwości (111)                                                                                    |  |
| Sloty (114)                                                                                          |  |
| Metaprogramowanie (114)                                                                              |  |
| Dekoratory jako metoda metaprogramowania (115)                                                       |  |
| Dekoratory klas (116)                                                                                |  |
| Wykorzystanie metody <code>__new__()</code> w celu nadpisania procesu tworzenia instancji klas (118) |  |
| Metaklasy (120)                                                                                      |  |
| Rady dotyczące automatycznego generowania kodu (127)                                                 |  |
| Podsumowanie (134)                                                                                   |  |
| <br>Rozdział 4. Właściwy dobór nazw (135)                                                            |  |
| PEP 8 i najlepsze praktyki nazewnicze (135)                                                          |  |
| Kiedy i dlaczego przestrzegać zasad PEP 8? (136)                                                     |  |
| Poza PEP 8 - wytyczne stylu w zespołach (136)                                                        |  |
| Notacje nazewnicze (137)                                                                             |  |
| Zmienne (138)                                                                                        |  |
| Zmienne publiczne i prywatne (140)                                                                   |  |
| Funkcje i metody (142)                                                                               |  |
| Właściwości (145)                                                                                    |  |
| Klasy (145)                                                                                          |  |
| Moduły i pakiety (146)                                                                               |  |
| Dobre praktyki nazewnicze (146)                                                                      |  |
| Użycie prefiksów <code>is</code> oraz <code>has</code> przy elementach logicznych (146)              |  |
| Użycie liczby mnogiej przy zmiennych przechowujących kolekcje (146)                                  |  |
| Precyzyjne opisywanie słowników (147)                                                                |  |
| Unikanie zbyt ogólnych określeń (147)                                                                |  |
| Unikanie istniejących nazw (148)                                                                     |  |
| Najlepsze praktyki dla argumentów funkcji i metod (149)                                              |  |
| Projektowanie argumentów metodą przyrostową (150)                                                    |  |
| Ufaj argumentom i testom (150)                                                                       |  |
| Ostrożne wykorzystanie magicznych argumentów <code>*args</code> oraz <code>**kwargs</code> (152)     |  |
| Nazwy klas (154)                                                                                     |  |
| Nazwy modułów i pakietów (154)                                                                       |  |
| Przydatne narzędzia (155)                                                                            |  |
| Pylint (155)                                                                                         |  |
| pep8 i flake8 (157)                                                                                  |  |
| Podsumowanie (158)                                                                                   |  |
| <br>Rozdział 5. Tworzenie i dystrybucja pakietów (159)                                               |  |
| Tworzenie pakietów (160)                                                                             |  |
| Zamieszanie wokół narzędzi do tworzenia i dystrybuowania pakietów (160)                              |  |
| Konfiguracja projektu (162)                                                                          |  |
| Własne polecenia skryptu <code>setup.py</code> (172)                                                 |  |
| Praca z pakietami podczas ich rozwoju (172)                                                          |  |

Pakiety przestrzeni nazw (174)

    Zastosowanie pakietów przestrzeni nazw (174)

    PEP 420 - domyślne pakiety przestrzeni nazw (176)

    Pakiety przestrzeni nazw w starszych wersjach Pythona (177)

Praca z repozytorium pakietów (178)

    Python Package Index - repozytorium pakietów Pythona (179)

    Dystrybucje źródłowe a dystrybucje budowane (181)

Samodzielne pliki wykonywalne (184)

    Kiedy samodzielne pliki wykonywalne są użyteczne? (186)

    Popularne narzędzia (186)

    Bezpieczeństwo kodu Pythona w samodzielnych plikach wykonywalnych (193)

Podsumowanie (195)

Rozdział 6. Zdalne wdrożenia kodu (197)

    Manifest Twelve-Factor App (198)

    Automatyzacja wdrożeń z wykorzystaniem narzędzia Fabric (200)

    Własne repozytorium pakietów lub kopie lustrzane PyPI (205)

        Utrzymywanie kopii lustrzanych PyPI (206)

        Wdrożenia z wykorzystaniem dystrybucji pakietów (207)

    Popularne konwencje i dobre praktyki (215)

        Hierarchia systemu plików (215)

        Izolacja (216)

        Wykorzystanie narzędzi nadzoru nad procesami (216)

        Kod aplikacji powinien być uruchomiony w przestrzeni użytkownika (218)

        Korzystanie ze wstecznych serwerów proxy protokołu HTTP (219)

        Przeładowywanie procesów bez zastoju (219)

    Instrumentacja i monitorowanie kodu (221)

        Logowanie błędów (Sentry oraz raven) (221)

        Monitorowanie metryk systemowych i aplikacji (224)

        Obsługa logów (226)

Podsumowanie (230)

Rozdział 7. Rozszerzenia Pythona w innych językach programowania (231)

    Inne języki, czyli C lub C++ (232)

        Jak działają rozszerzenia w C i C++ (232)

    Dlaczego warto tworzyć rozszerzenia (234)

        Zwiększanie wydajności w krytycznych sekcjach kodu (235)

        Integracja kodu napisanego w innych językach programowania (236)

        Integracja zewnętrznych bibliotek dynamicznych (236)

        Tworzenie własnych wbudowanych typów danych (236)

    Pisanie rozszerzeń (237)

        Zwyczajne rozszerzenia w C (238)

        Cython (253)

    Wyzwania związane z rozszerzeniami (257)

        Dodatkowa złożoność (258)

        Debugowanie (258)

    Korzystanie z dynamicznych bibliotek bez pisania rozszerzeń (259)

        ctypes (259)

        CFFI (265)

Podsumowanie (266)

Rozdział 8. Zarządzanie kodem (267)

## Systemy kontroli wersji (268)

- Scentralizowane systemy kontroli wersji (268)

- Rozproszone systemy kontroli wersji (271)

- Systemy scentralizowane czy rozproszone? (274)

- Korzystaj z systemu Git, jeśli tylko możesz (274)

- Git flow oraz GitHub flow (275)

## Ciągłe procesy programistyczne (279)

- Ciągła integracja oprogramowania (280)

- Ciągłe dostarczanie oprogramowania (284)

- Ciągłe wdrażanie oprogramowania (285)

- Popularne narzędzia do ciągłej integracji (285)

- Wybór odpowiednich narzędzi i częste pułapki (294)

## Podsumowanie (297)

## Rozdział 9. Dokumentowanie projektu (299)

### Siedem zasad technicznego pisanie (300)

- Pisz w dwóch krokach (300)

- Skieruj przekaz do konkretnej grupy czytelników (301)

- Korzystaj z prostego stylu (302)

- Ogranicz zakres informacji (303)

- Korzystaj z realistycznych przykładów (303)

- Dokumentuj lekko, ale jednocześnie wystarczająco (304)

- Korzystaj z szablonów (305)

### Poradnik reStructuredText (305)

- Struktura sekcji (307)

- Listy numerowane i wypunktowania (309)

- Formatowanie znakowe (310)

- Bloki dosłowne (310)

- Odnośniki (311)

### Budowanie dokumentacji (312)

- Budowanie portfolio dokumentacji (312)

### Tworzenie własnego portfolio (319)

- Projektowanie krajobrazu dokumentacji (319)

- Budowanie dokumentacji a systemy ciągłej integracji (324)

### Podsumowanie (325)

## Rozdział 10. Programowanie sterowane testami (327)

### Nie testuję (327)

- Zasady programowania sterowanego testami (328)

- Możliwe rodzaje testów (332)

- Narzędzia testowe standardowej biblioteki Pythona (335)

### Testuję (340)

- Pułapki modułu unittest (340)

- Alternatywy dla modułu unittest (341)

- Mierzenie pokrycia kodu testami (349)

- Fałszywe obiekty zastępcze i atrapy (351)

- Testowanie kompatybilności środowisk i zależności (358)

- Programowanie sterowane dokumentami (361)

### Podsumowanie (363)

## Rozdział 11. Optymalizacja - ogólne zasady i techniki profilowania (365)

### Trzy zasady optymalizacji (365)

- Przede wszystkim spraw, aby kod działał poprawnie (366)
- Pracuj z perspektywy użytkownika (367)
- Utrzymuj kod czytelnym (367)
- Strategia optymalizacyjna (368)
  - Poszukaj innego winowajcy (368)
  - Skaluj sprzęt (369)
  - Napisz test wydajnościowy (370)
- Identyfikowanie wąskich gardeł wydajności (370)
  - Profilowanie czasu użycia procesora (370)
  - Profilowanie zużycia pamięci (379)
  - Profilowanie połączeń sieciowych (389)
- Podsumowanie (390)

## Rozdział 12. Optymalizacja - wybrane skuteczne techniki (391)

- Redukcja złożoności (392)
  - Złożoność cyklomatyczna (394)
  - Notacja dużego O (394)
- Upraszczenie (397)
  - Przeszukiwanie list (397)
  - Korzystanie ze zbiorów w miejscu list (398)
  - Ukróć zewnętrzne wywołania, zredukuj nakład pracy (398)
  - Korzystanie z modułu collections (399)
- Stosowanie kompromisów architektonicznych (403)
  - Stosowanie heurystyk i algorytmów aproksymacyjnych (403)
  - Stosowanie kolejek zadań i opóźnionego przetwarzania (404)
  - Stosowanie probabilistycznych struktur danych (408)
- Buforowanie (409)
  - Buforowanie deterministyczne (410)
  - Buforowanie niedeterministyczne (412)
  - Usługi buforujące (413)
- Podsumowanie (416)

## Rozdział 13. Przetwarzanie współbieżne i równoległe (419)

- Dlaczego współbieżność? (420)
- Wielowątkowość (421)
  - Czym jest wielowątkowość? (422)
  - Jak Python radzi sobie z wątkami (423)
  - Kiedy należy korzystać z wielowątkowości? (424)
- Przetwarzanie wieloprotokółowe (439)
  - Wbudowany moduł multiprocessing (442)
- Programowanie asynchroniczne (447)
  - Kooperacyjna wielozadaniowość i asynchroniczne operacje wejścia/wyjścia (448)
  - Słowa kluczowe async i await (449)
  - asyncio w starszych wersjach Pythona (453)
  - Praktyczny przykład programu asynchronicznego (454)
  - Integracja nieasynchronicznego kodu z async za pomocą modułu futures (456)
- Podsumowanie (459)

## Rozdział 14. Przydatne wzorce projektowe (461)

- Wzorce kreacyjne (462)
  - Singleton (462)
- Wzorce strukturalne (465)

- Adapter (466)
- Pełnomocnik (481)
- Fasada (482)
- Wzorce czynnościowe (483)
  - Obserwator (483)
  - Odwiedzający (485)
  - Szablon (488)
- Podsumowanie (491)
- Skorowidz (492)