Tworzenie własnej biblioteki programistycznej w Pythonie

lab 3

1 Wprowadzenie teoretyczne

Tworzenie biblioteki programistycznej (tzw. pakietu) w języku Python polega na zorganizowaniu kodu w taki sposób, by można go było łatwo wykorzystywać w wielu projektach. Dzięki temu unikamy duplikacji kodu, ułatwiamy konserwację i rozwijanie oprogramowania. Biblioteki mogą zawierać funkcje, klasy, moduły i całe pakiety, które realizują określoną funkcjonalność (np. obliczenia numeryczne, przetwarzanie tekstu, obsługę plików).

Istotnymi elementami biblioteki w Pythonie są:

- Struktura katalogów i plików (zawierająca moduły i plik `__init__.py`),
- Pliki `setup.py` lub `pyproject.toml` (jeśli planuje się dystrybucję i instalację pakietu),
- Dokumentacja w postaci docstringów oraz plików (np. `README.md`),
- Testy jednostkowe potwierdzające poprawność działania biblioteki.

2 Cel zadania

1. Zapoznanie się z koncepcją tworzenia własnej biblioteki programistycznej w języku Python.
2. Zrozumienie struktury pakietów i modułów w Pythonie.
3. Przygotowanie pakietu do wielokrotnego wykorzystania w różnych projektach.
4. Utrwalenie praktyk związanych z dokumentowaniem kodu, testowaniem oraz kontrolą jakości.

3 Zakres zadania

1. Stworzenie struktury katalogów i plików dla biblioteki (co najmniej jeden pakiet i kilka modułów).
2. Implementacja przynajmniej trzech modułów w ramach biblioteki, zawierających w sumie co najmniej 6–8 funkcji lub klas o wyraźnie zdefiniowanej funkcjonalności.
3. Dbanie o zgodność z konwencjami PEP 8 (kod powinien być czytelny i łatwy do utrzymania).
4. Zaimplementowanie testów jednostkowych (przynajmniej po kilka testów na moduł).
5. Uwzględnienie dokumentacji kodu w docstringach i przygotowanie pliku `README.md` opisującego główne funkcjonalności biblioteki.
6. (Opcjonalnie) Przygotowanie pliku `setup.py` lub `pyproject.toml` w celu ułatwienia instalacji i dystrybucji biblioteki.
7. Publikacja biblioteki w repozytorium GitHub.

4 Instrukcje – krok po kroku

4.1 Struktura biblioteki

1. Utwórz folder główny projektu, np. `my_awesome_lib`.
2. Wewnątrz utwórz plik `__init__.py`, aby Python traktował folder jako pakiet.
3. Dodaj co najmniej trzy moduły (pliki `.py`) w tym folderze, np. `data_utils.py`, `math_tools.py`, `text_processing.py`.
4. (Opcjonalnie) Jeśli chcesz rozbić bibliotekę na wiele pakietów, utwórz podfoldery i umieść w nich swoje moduły z plikami `__init__.py`.

4.2 Implementacja funkcjonalności

1. Zaprojektuj funkcje lub klasy w każdym module tak, by rozwiązywały konkretne problemy (np. operacje na ciągach znaków, konwersje danych, obliczenia matematyczne, itd.).

2. Staraj się, by kod był podzielony logicznie: każda funkcja powinna mieć jeden wyraźny cel.

3. Dodaj komentarze i docstringi dokumentujące sposób użycia każdej funkcji (argumenty, typ zwracany, obsługiwane wyjątki).

4.3 Testy jednostkowe i jakość kodu

1. W folderze `tests` (na tym samym poziomie co `my_awesome_lib`) utwórz pliki testowe, np. `test_data_utils.py`, `test_math_tools.py`.

2. Każdy plik testowy powinien zawierać zestaw testów jednostkowych (`unittest` lub `pytest`).

3. Napisz testy sprawdzające prawidłowe działanie funkcji w różnych scenariuszach (w tym przypadki brzegowe i błędne).

4. Sprawdź, czy kod przechodzi testy przed publikacją.

5. (Opcjonalnie) Użyj narzędzi do lintingu (np. `flake8`, `pylint`) oraz formatowania (np. `black`) w celu zachowania spójnego stylu.

4.4 Dokumentacja i plik README

1. Przygotuj plik `README.md` w głównym folderze projektu, zawierający:

   - Krótki opis biblioteki i jej głównych możliwości.
   - Instrukcję instalacji (ręczna lub przez `pip install -e .`, jeśli dostępny jest `setup.py`).
   - Przykłady wywołań funkcji, jeśli to możliwe.
   - Informację o licencji, autorze i wersji.

2. (Opcjonalnie) Rozważ użycie docstringów generujących dokumentację w formacie Sphinx lub MkDocs.

4.5 Publikacja biblioteki na GitHub

1. Utwórz nowe repozytorium na GitHub.

2. Zainicjuj repozytorium lokalnie w folderze projektu (`git init`).

3. Dodaj, commituj i wypchnij (`push`) kod do głównej gałęzi repozytorium.

4. Upewnij się, że w pliku `.gitignore` znajdują się wpisy wykluczające zbędne pliki (np. `__pycache__`, `.pytest_cache`, pliki środowiskowe).

5. (Opcjonalnie) Skonfiguruj GitHub Actions do automatycznego uruchamiania testów przy każdym pushu.

5 Kryteria oceny

1. Struktura pakietu i modułów (poprawna organizacja, stosowanie `__init__.py`).

2. Kompletność i jakość kodu (co najmniej 3 moduły, 6–8 funkcji lub klas, czytelność i zgodność z PEP 8).

3. Testy jednostkowe obejmujące kluczowe funkcje (przynajmniej kilka testów dla każdego modułu).

4. Obecność i jakość dokumentacji (docstringi, README, licencja).

5. Publikacja w repozytorium GitHub (poprawna konfiguracja, widoczność kodu).

6. (Opcjonalnie) Dodatkowe punkty za integrację CI/CD, wysokie pokrycie testowe lub inne rozwiązania usprawniające pracę z biblioteką.