Testowanie aplikacji Test-Driven Development (TDD)

lab 2

1 Cel zadania

1. Zapoznanie się z podstawami testowania aplikacji w Pythonie.
2. Nauka podejścia Test-Driven Development (TDD).
3. Implementacja testów jednostkowych z wykorzystaniem modułu `unittest`.
4. Wykorzystanie testów do poprawy jakości kodu.
5. Zapoznanie się z narzędziami do analizy pokrycia testowego.

2 Zakres zadania

1. Implementacja testów jednostkowych w Pythonie.
2. Praca w podejściu Test-Driven Development.
3. Zaprojektowanie i zaimplementowanie pięciu funkcji o ustalonej, jasno zdefiniowanej funkcjonalności – powinny być zróżnicowane (np. operacje na ciągach znaków, przetwarzanie list, proste obliczenia matematyczne, zarządzanie danymi itd.).
4. Modyfikacja kodu w celu spełnienia testów.
5. Zapoznanie się z zaawansowanymi technikami testowania, takimi jak testy parametryzowane.
6. Analiza pokrycia kodu testami.
7. Publikacja wyników na GitHub i złożenie zadania na Moodle.

3 Instrukcje – krok po kroku

3.1 Utworzenie środowiska testowego

1. Utwórz katalog `zadanie_2` w repozytorium GitHub.
2. W katalogu stwórz plik `test_app.py`, w którym będą znajdować się testy.
3. Utwórz plik `app.py`, w którym zostaną zaimplementowane funkcje.
4. Sprawdź, czy masz zainstalowany moduł `unittest` (jest dostępny w standardowej bibliotece Pythona).
5. Zainstaluj dodatkowe narzędzia, jeśli planujesz korzystać z `pytest` lub `coverage.py`.

3.2 Funkcje do zaimplementowania i przetestowania

W pliku `app.py` należy zaimplementować pięć funkcji o ustalonej, jasno zdefiniowanej funkcjonalności. Przykładowe pomysły:

- Funkcja sprawdzająca poprawność adresu e-mail.
- Funkcja dokonująca prostych obliczeń matematycznych (np. obliczanie pola figury).
- Funkcja przetwarzająca listę danych (np. filtracja, sortowanie).
- Funkcja konwertująca format dat.
- Funkcja sprawdzająca, czy tekst jest palindromem.

Każda funkcja powinna mieć jasno określony cel i parametry wejściowe/wyjściowe.

3.3 Test-Driven Development – cykl pracy

1. Napisanie testu – przed napisaniem kodu, najpierw tworzymy testy sprawdzające oczekiwane zachowanie.
2. Uruchomienie testów – testy powinny na początku nie przechodzić (ponieważ kod nie jest jeszcze napisany).
3. Implementacja funkcji – tworzymy minimalny kod, który spełnia testy.

4. Uruchomienie testów ponownie – sprawdzamy, czy testy przechodzą.
5. Refaktoryzacja – jeśli kod działa poprawnie, można go zoptymalizować.
6. Analiza pokrycia testowego – sprawdzamy, jakie fragmenty kodu nie zostały przetestowane.

3.4 Implementacja testów w Pythonie

- Każda z pięciu funkcji powinna mieć co najmniej trzy testy jednostkowe, uwzględniające typowe przypadki, przypadki brzegowe oraz błędne dane wejściowe (jeśli dotyczy).
- Korzystaj z metod asercji dostarczanych przez `unittest` (np. `assertEqual`, `assertTrue`, `assertRaises`, itd.).
- Dla uzyskania lepszej organizacji testów, skorzystaj z klasy dziedziczącej po `unittest.TestCase`.

3.4.1 Ocena 3.0 – Minimalne wymagania

1. Implementacja testów dla co najmniej jednej funkcji.
2. Użycie podstawowych asercji, np. `assertEqual()`.
3. Uruchomienie testów i sprawdzenie ich działania.

3.4.2 Ocena 4.0 – Poprawne i rozszerzone testy

1. Dodanie kilku testów dla różnych przypadków brzegowych.
2. Wykorzystanie większej liczby metod testowych, np. `assertNotEqual()`, `assertRaises()`.
3. Organizacja testów w osobnej klasie dziedziczącej po `unittest.TestCase`.
4. Poprawne zarządzanie błędami i wyjątkami w kodzie testowanym.
5. Implementacja testów parametryzowanych z wykorzystaniem `unittest` lub `pytest.mark.parametrize`.

3.4.3 Ocena 5.0 – Pełne pokrycie testowe i najlepsze praktyki

1. Stworzenie testów dla wszystkich zaimplementowanych funkcji (co najmniej 5).
2. Zapewnienie pełnego pokrycia testowego dla wszystkich przypadków.
3. Wykorzystanie metody `setUp()` do przygotowania środowiska testowego.
4. Implementacja testów parametryzowanych.
5. Dodanie komentarzy do kodu testowego, wyjaśniających testowane przypadki.
6. Sprawdzenie poprawności testów za pomocą narzędzia do mierzenia pokrycia kodu, np. `coverage.py`.

3.5 Publikacja kodu na GitHub

1. Wykonaj commit zmian i dodaj opis (np. „Dodano testy jednostkowe").
2. Wypchnij kod do repozytorium GitHub (`git push origin main`).
3. Sprawdź, czy kod jest widoczny w przeglądarce GitHub.

4 Kryteria oceny

Ocena uzależniona jest od kompletności przesłanego zadania, jakości testów i ich poprawności. Czas na wykonanie 7 dni.