Погружение в Python обучение в записи Урок 28

Задание 1. Тестирование класса с использованием pytest

Напишите класс `BankAccount`, который управляет балансом счета. Он должен поддерживать следующие методы:

- `deposit(amount)`: добавляет указанную сумму к балансу.
- `withdraw(amount)`: снимает указанную сумму с баланса, если достаточно средств.
- `get_balance()`: возвращает текущий баланс счета.

При попытке снять больше средств, чем доступно на счете, должно выбрасываться исключение `InsufficientFundsError`. Напишите как минимум 5 тестов для проверки работы классов и его методов.

Подсказка № 1

Проверьте, что начальный баланс создаваемого объекта `BankAccount` корректен, используя значение, переданное в конструкторе. Убедитесь, что баланс инициализируется правильно при создании объекта.

Подсказка № 2

Проверьте, что метод `deposit` корректно добавляет указанную сумму к текущему балансу. Убедитесь, что сумма депозита положительна и увеличивает баланс на ожидаемое значение.

Подсказка № 3

Убедитесь, что метод `withdraw` корректно уменьшает баланс на указанную сумму, если на счету достаточно средств. Проверьте правильность работы метода при различных значениях суммы.

Подсказка № 4

Убедитесь, что метод `withdraw` корректно выбрасывает исключение `InsufficientFundsError`, когда пытаются снять больше средств, чем доступно на счету. Используйте `pytest.raises` для проверки этого поведения.

Подсказка № 5

Проверьте, что при создании объекта BankAccount без указания начального баланса, баланс инициализируется как 0. Это поможет убедиться в правильности работы конструктора с дефолтными значениями.

Эталонное решение:

```
class InsufficientFundsError(Exception):
    def __init__(self):
        super().__init__("Недостаточно средств на счете.")

class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Сумма депозита должна быть положительной.")
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            raise InsufficientFundsError()
        self.balance -= amount

    def get_balance(self):
        return self.balance
```

Код тестирования:

```
import pytest


@pytest.fixture
def bank_account():
    # Подготовка тестового состояния
    return BankAccount(100)  # Начальный баланс 100


def test_initial_balance(bank_account):
    # Проверка начального баланса
    assert bank_account.get_balance() == 100


def test_deposit(bank_account):
    # Проверка депозита
    bank_account.deposit(50)
    assert bank_account.get_balance() == 150


def test_withdraw(bank_account):
    # Проверка снятия средств
    bank_account.withdraw(30)
    assert bank_account.get_balance() == 70


def test_withdraw_insufficient_funds(bank_account):
    # Проверка снятия больше средств, чем доступно
    with pytest.raises(InsufficientFundsError):
        bank_account.withdraw(200)
```

```
def test_deposit_negative_amount(bank_account):
    # Проверка депозита отрицательной суммы
    with pytest.raises(ValueError):
        bank_account.deposit(-10)
```

Задача 2. Тестирование класса с использованием `unittest`

Напишите класс `Library`, который управляет книгами. Класс должен поддерживать следующие методы:

- `add_book(title)`: добавляет книгу в библиотеку.
- `remove_book(title)`: удаляет книгу из библиотеки.
- `list_books()`: возвращает список всех книг в библиотеке.

При попытке удалить книгу, которая не существует, должно выбрасываться исключение `BookNotFoundError`. Для тестирования используйте unitest.

Подсказка № 1

Убедитесь, что метод `add_book` корректно добавляет книги в библиотеку. Для этого добавьте книгу и проверьте, что она присутствует в списке всех книг.

Подсказка № 2

Проверьте, что метод `remove_book` корректно удаляет книгу из библиотеки. Добавьте книгу, удалите ее и убедитесь, что она больше не присутствует в списке.

Подсказка № 3

Убедитесь, что метод `remove_book` корректно выбрасывает исключение `BookNotFoundError`, если пытаетесь удалить книгу, которой нет в библиотеке.

Подсказка № 4

Проверьте, что метод list_books возвращает правильный список книг после выполнения нескольких операций добавления и удаления книг.

Эталонное решение:

```
class BookNotFoundError(Exception):
    def __init__(self):
        super().__init__("Книга не найдена в библиотеке.")


class Library:
    def __init__(self):
        self.books = set()


    def add_book(self, title):
        self.books.add(title)


    def remove_book(self, title):
        if title not in self.books:
            raise BookNotFoundError()
        self.books.remove(title)


    def list_books(self):
        return list(self.books)
```

Код тестирования:

```
import unittest


class TestLibrary(unittest.TestCase):


    def setUp(self):

        self.library = Library()


    def test_add_book(self):

        self.library.add_book("1984")

        self.assertIn("1984", self.library.list_books())


    def test_remove_book(self):

        self.library.add_book("Brave New World")

        self.library.remove_book("Brave New World")

        self.assertNotIn("Brave New World",
self.library.list_books())


    def test_remove_nonexistent_book(self):

        with self.assertRaises(BookNotFoundError):

            self.library.remove_book("Nonexistent Book")


if __name__ == '__main__':

    unittest.main()
```

Задача 3. Тестирование класса с использованием `doctest`

Напишите класс `Rectangle`, который управляет прямоугольником. Класс должен поддерживать следующие методы:

- `set_dimensions(width, height)`: устанавливает ширину и высоту прямоугольника.
- `get_area()`: возвращает площадь прямоугольника.
- `get_perimeter()`: возвращает периметр прямоугольника.

Напишите 3 теста с помощью doctest.

Подсказка № 1

Убедитесь, что при создании объекта `Rectangle` с заданными шириной и высотой, методы `get_area` и `get_perimeter` возвращают правильные значения.

Подсказка № 2

После установки новых размеров с помощью `set_dimensions`, убедитесь, что методы `get_area` и `get_perimeter` обновляются корректно.

Подсказка № 3

Убедитесь, что метод `set_dimensions` выбрасывает исключение `ValueError`, если переданы отрицательные значения для ширины или высоты.

Подсказка № 4

Убедитесь, что метод `set_dimensions` правильно обрабатывает нулевые значения. Для нулевых значений площадь и периметр должны быть корректными (площадь будет 0).

Эталонное решение:

```
class Rectangle:

    def __init__(self, width=0, height=0):

        """

        Инициализация прямоугольника с заданными шириной и высотой.



        >>> r = Rectangle(3, 4)

        >>> r.get_area()
```

```
        12
        >>> r.get_perimeter()
        14
        """
        self.width = width
        self.height = height

    def set_dimensions(self, width, height):
        """
        Устанавливает ширину и высоту прямоугольника.

        >>> r = Rectangle()
        >>> r.set_dimensions(6, 7)
        >>> r.get_area()
        42
        >>> r.get_perimeter()
        26
        """
        if width <= 0 or height <= 0:
            raise ValueError("Ширина и высота должны быть
положительными числами.")
        self.width = width
        self.height = height

    def get_area(self):
        """Возвращает площадь прямоугольника."""
        return self.width * self.height
```

```
    def get_perimeter(self):
        """Возвращает периметр прямоугольника."""
        return 2 * (self.width + self.height)


import doctest


if __name__ == "__main__":
    doctest.testmod()
```