Python Testing Lab with Pytest

In this lab, we will practice writing tests for a Python class using `pytest`. We'll focus on basic testing, fixtures, and parameterized tests.

Part 1: BankAccount Class

We will start with the `BankAccount` class. Here is the implementation you will be testing:

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

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
        else:
            raise ValueError("Deposit amount must be positive")

    def withdraw(self, amount):
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        elif amount <= 0:
            raise ValueError("Withdrawal amount must be positive")
        else:
            self.balance -= amount

    def get_balance(self):
        return self.balance

    def reset_account(self):
        self.balance = 0
```

Part 2: Writing Basic Tests

Create a file named `test_bank_account.py`. Implement the following tests:

Test Initial Balance

Objective: Verify the initial balance of a new account is 0.

```
def test_initial_balance():
    # Hint: Create an instance of BankAccount and check the balance
    pass
```

Test Deposit

Objective: Verify depositing money increases the balance correctly.

```
def test_deposit():
    # Hint: Deposit an amount and check the balance
    pass
```

Test Withdraw

Objective: Verify withdrawing money decreases the balance correctly.

```
def test_withdraw():
    # Hint: Deposit an amount, then withdraw a smaller amount, and check the balance
    pass
```

Test Withdraw Insufficient Funds

Objective: Verify withdrawing more than the balance raises an error.

```
def test_withdraw_insufficient_funds():
    # Hint: Attempt to withdraw without depositing and expect a ValueError
    pass
```

Test Negative Deposit

Objective: Verify depositing a negative amount raises an error.

```
def test_negative_deposit():
    # Hint: Attempt to deposit a negative amount and expect a ValueError
    pass
```

Test Negative Withdraw

Objective: Verify withdrawing a negative amount raises an error.

```
def test_negative_withdraw():
    # Hint: Attempt to withdraw a negative amount and expect a ValueError
    pass
```

Test Reset Account

Objective: Verify resetting the account sets the balance back to 0.

```
def test_reset_account():
    # Hint: Deposit an amount, reset the account, and check the balance
    pass
```

Part 3: Using Fixtures

Fixtures are useful for setting up preconditions and shared state for your tests.

Create a Fixture

In your `test_bank_account.py` file, create a fixture for a `BankAccount` instance:

```
import pytest

@pytest.fixture
def bank_account():
    return BankAccount()
```

Update Tests to Use Fixture

Update your tests to use the `bank_account` fixture:

```
def test_initial_balance(bank_account):
    # Hint: Use the bank_account fixture
    pass
```

Repeat for the other tests, using the `bank_account` fixture.

Part 4: Parameterized Tests

Parameterized tests allow you to run a test function with different sets of inputs and expected outputs.

Parameterize Test Deposit

Objective: Test depositing various amounts using parameterized tests.

```python
@pytest.mark.parametrize("initial, deposit_amount, expected", [
    (0, 100, 100),
    (0, 50, 50),
    (0, 0, 0),
])
def test_deposit(bank_account, initial, deposit_amount, expected):
    # Hint: Deposit different amounts and check the balance
    pass
```

Parameterize Test Withdraw

Objective: Test withdrawing various amounts using parameterized tests.

```python
@pytest.mark.parametrize("initial, deposit_amount, withdraw_amount, expected", [
    (0, 100, 50, 50),
    (0, 50, 25, 25),
    (0, 200, 200, 0),
])
def test_withdraw(bank_account, initial, deposit_amount, withdraw_amount, expected):
    # Hint: Deposit then withdraw different amounts and check the balance
    pass
```

Parameterize Exception Tests

Objective: Test scenarios that should raise exceptions using parameterized tests.

```python
@pytest.mark.parametrize("initial, deposit_amount, withdraw_amount, exception, message", [
    (0, 0, 50, ValueError, "Insufficient funds"),
    (0, 100, -10, ValueError, "Withdrawal amount must be positive"),
    (0, -10, 0, ValueError, "Deposit amount must be positive"),
])
def test_exceptions(bank_account, initial, deposit_amount, withdraw_amount, exception, message):
    # Hint: Use parameterized inputs to test for exceptions
    pass
```