04/27/2025
Logan Gray

Assignment 4: Testing

Part 1: Unit Testing (20 points)

- Write unit tests for at least 5 key functions using pytest

Tests can be found in tests/test_basic.py

- Achieve at least 90% code coverage

Can be verified by running “$ pytest --cov=src” in the root directory

- Document your approach to unit testing

In order to achieve 90% coverage, I decided I needed to write a unit test for each of the functions in tasks.py. For each function, I read and understood the code and tried to come up with test cases for each. I wanted enough cases so that all branches of code in the function would be run by at least on of the tests, and I tried to think of edge cases for each function and included cases for all possible types of input. For example, for generate_unique_id(), I made sure my cases covered when the passed list of tasks was empty and when the list of tasks had non-contiguous IDs. The general flow for the unit tests was to set up the data I needed, pass it into the function, and assert the result is what I expected.

- Have a streamlit button. On button press it runs the tests

This button can be found at the top of the page.

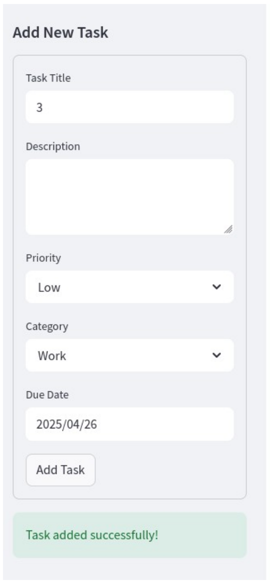
Part 2: Bug Reporting and Fixing (20 points)

- Document all bugs found using a standard bug report format
- Fix all bugs you've identified
- Provide before/after evidence of fixes

Here I will provide documentation of bugs I found with before and after evidence of the fixes. These fixes are implemented in the submitted repository.

Bug 1: Adding Task Throws Error

Behavior: Adding a task after deleting a task (that is not the most recently added) throws a duplicate element key error:



Add New Task

Task Title: 3

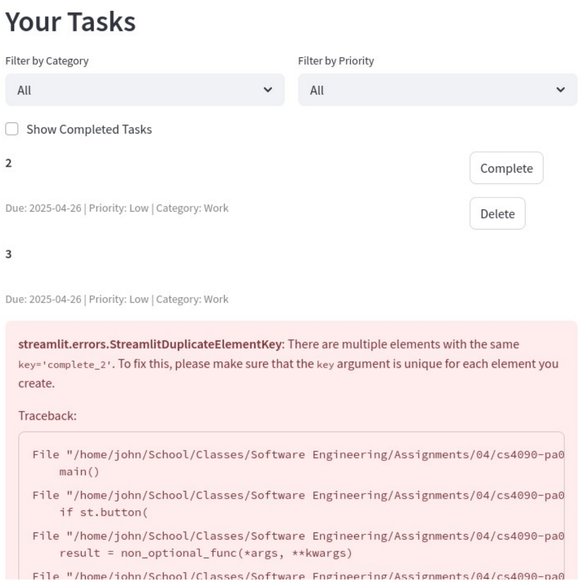
Description:

Priority: Low

Category: Work

Due Date: 2025/04/26

Task added successfully!



Your Tasks

Filter by Category: All

Filter by Priority: All

2

Due: 2025-04-26 | Priority: Low | Category: Work

3

Due: 2025-04-26 | Priority: Low | Category: Work

streamlit.errors.StreamlitDuplicateElementKey: There are multiple elements with the same `key='complete_2'`. To fix this, please make sure that the `key` argument is unique for each element you create.

Traceback:

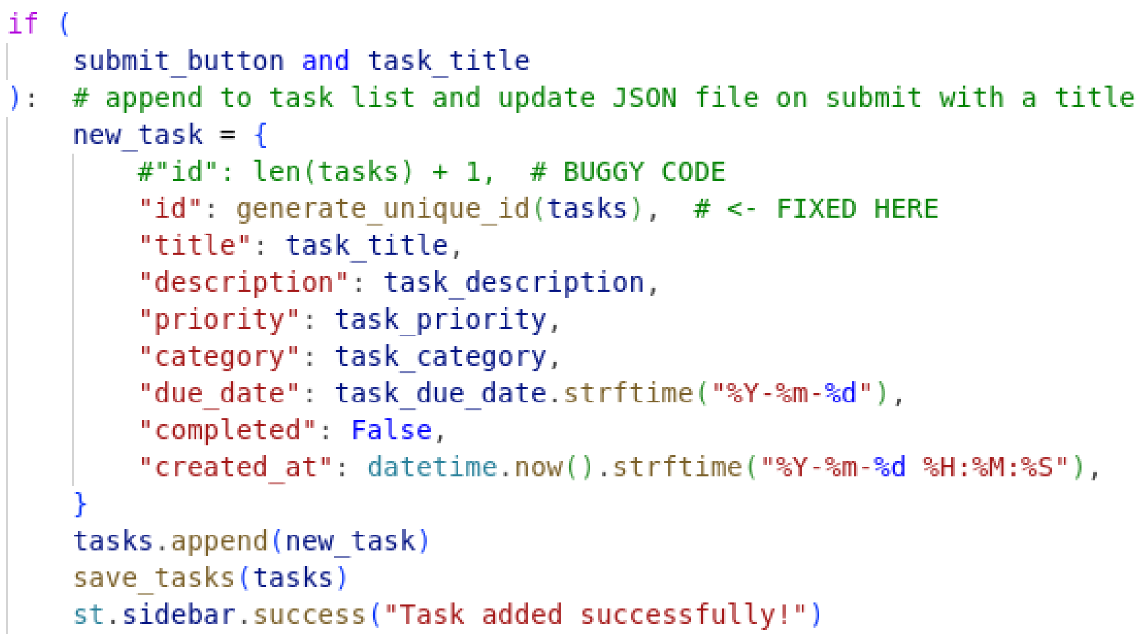
```
File "/home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa0
    main()
File "/home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa0
    if st.button(
File "/home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa0
    result = non_optional_func(*args, **kwargs)
File "/home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa0
```

Expected Behavior: Adding a task appends it to the task list with no errors and the display is updated accordingly

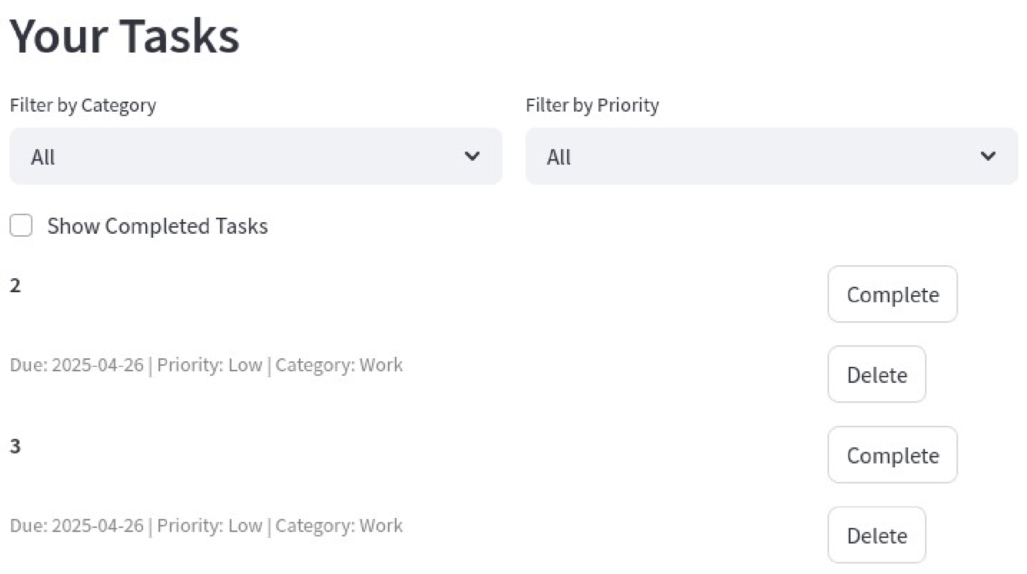
Steps to Reproduce:

1. Start with empty task list
2. Add Task 1
3. Add Task 2
4. Delete Task 1
5. Add Task 3 and an error will be thrown

Fix: This bug occurred because app.py did not correctly implement the logic for task ID generation when appending a task. Before the fix, new task IDs were generated with “len(tasks) + 1” which is not correct. Here a call should have been made to generate_unique_id() from tasks.py. I fixed the bug by replacing the incorrect logic with the appropriate function call:



```python
if (
    submit_button and task_title
):  # append to task list and update JSON file on submit with a title
    new_task = {
        #"id": len(tasks) + 1,  # BUGGY CODE
        "id": generate_unique_id(tasks),  # <- FIXED HERE
        "title": task_title,
        "description": task_description,
        "priority": task_priority,
        "category": task_category,
        "due_date": task_due_date.strftime("%Y-%m-%d"),
        "completed": False,
        "created_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    }
    tasks.append(new_task)
    save_tasks(tasks)
    st.sidebar.success("Task added successfully!")
```



Your Tasks

Filter by Category
All

Filter by Priority
All

Show Completed Tasks

2

Due: 2025-04-26 | Priority: Low | Category: Work

3

Due: 2025-04-26 | Priority: Low | Category: Work

Bug 2: Tasks without Due Date Considered Overdue

Behavior: get_overdue_tasks() included tasks with no due date in the returned list

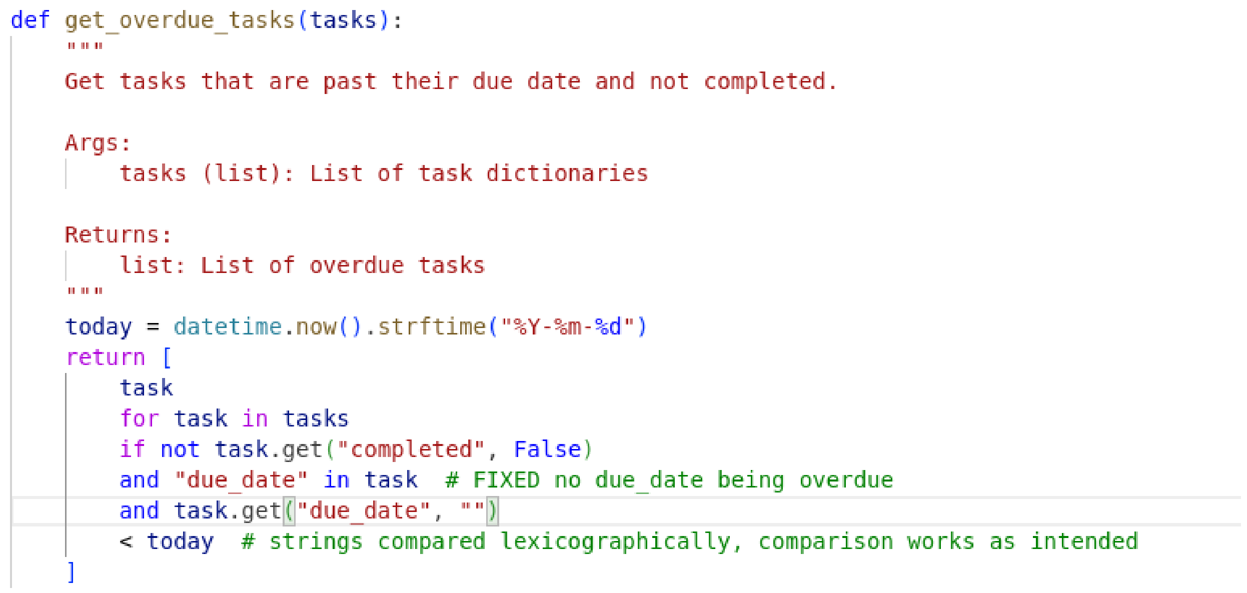
Expected Behavior: Tasks with no due date should not be included because they cannot be “overdue”

Steps to Reproduce:

1. Construct a list of tasks including at least one task without a “due_date” key.

2. Pass the list into get_overdue_tasks()
3. Observe the returned list contains all entries without “due_date” keys

Fix: This happens because the function compares the due date to the current day with the < operator. If there is no due_date, the empty string is returned which is less than all dates and the task is considered over due. I fixed this by adding another condition to the boolean statement that checks if the task has a due date:



```python
def get_overdue_tasks(tasks):
    """
    Get tasks that are past their due date and not completed.

    Args:
        tasks (list): List of task dictionaries

    Returns:
        list: List of overdue tasks
    """
    today = datetime.now().strftime("%Y-%m-%d")
    return [
        task
        for task in tasks
        if not task.get("completed", False)
        and "due_date" in task  # FIXED no due_date being overdue
        and task.get("due_date", "")
        < today  # strings compared lexicographically, comparison works as intended
    ]
```

Part 3: Pytest Features (20 points)

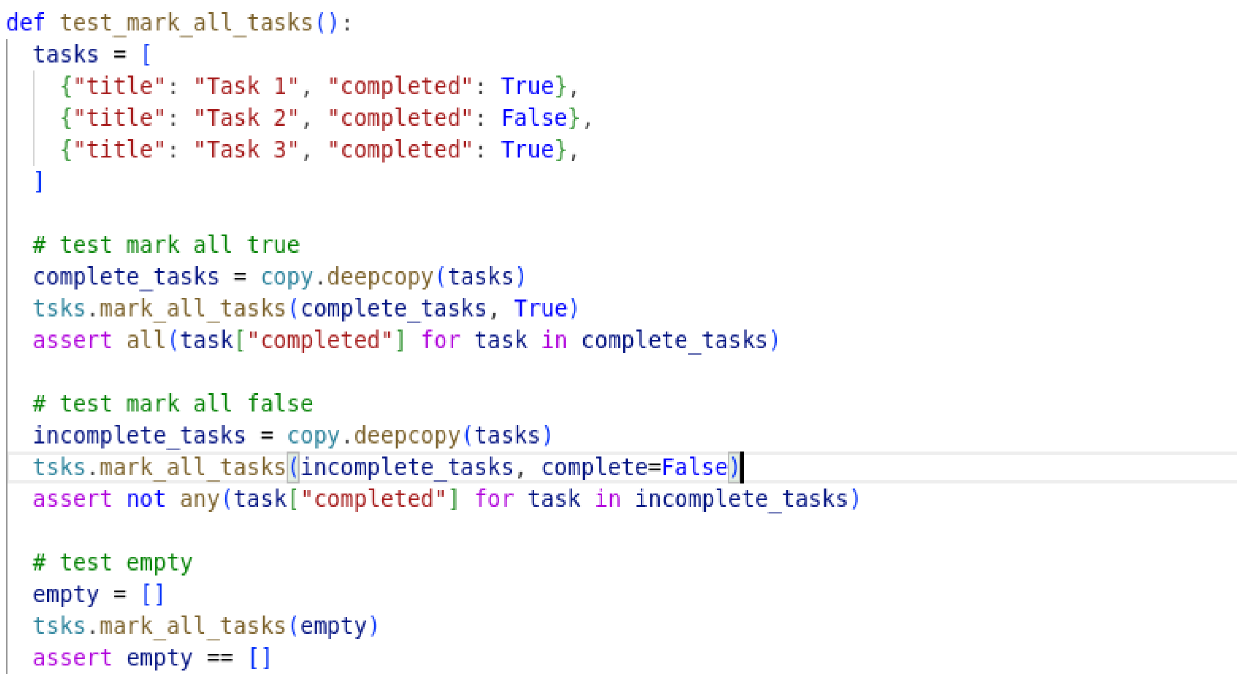
For this part I populated the test_advanced.py file with unit tests that utilize fixtures and parametrization for the tasks.py functions. For each unit test, I came up with multiple test cases and added them as parameters. This was convenient because I didn't have to have a separate assert call for each case.
Then, I added buttons to my streamlit app to showcase certain pytest functionalities. I added buttons for:

- Run Parameterized Tests
- Check Test Coverage
- Generate HTML Report
- Run with Mock

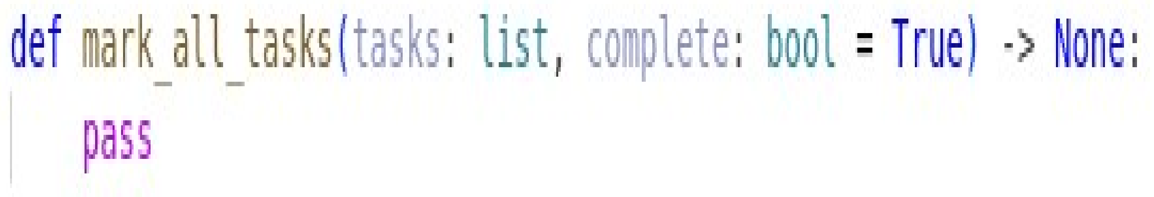
Part 4: Do Test-Driven Development (TDD) (20 points)

Feature 1: Mark All Tasks Complete/Incomplete

- Initial test creation

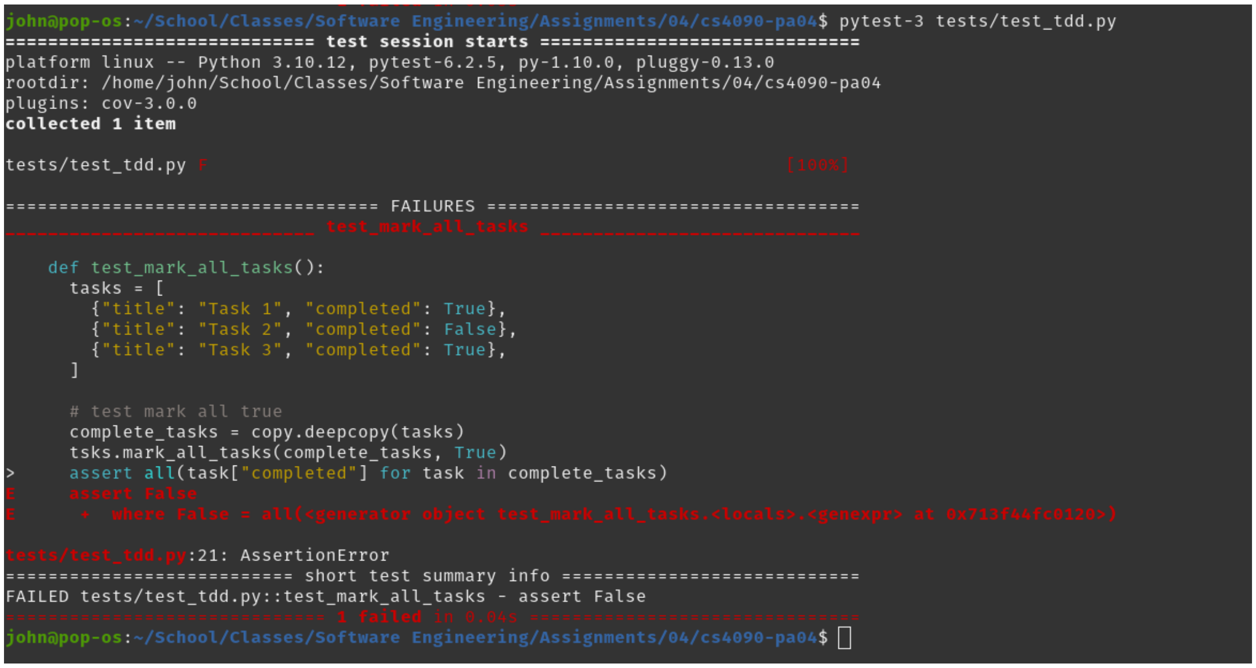


```python
def test_mark_all_tasks():
    tasks = [
        {"title": "Task 1", "completed": True},
        {"title": "Task 2", "completed": False},
        {"title": "Task 3", "completed": True},
    ]

    # test mark all true
    complete_tasks = copy.deepcopy(tasks)
    tsks.mark_all_tasks(complete_tasks, True)
    assert all(task["completed"] for task in complete_tasks)

    # test mark all false
    incomplete_tasks = copy.deepcopy(tasks)
    tsks.mark_all_tasks(incomplete_tasks, complete=False)
    assert not any(task["completed"] for task in incomplete_tasks)

    # test empty
    empty = []
    tsks.mark_all_tasks(empty)
    assert empty == []
```



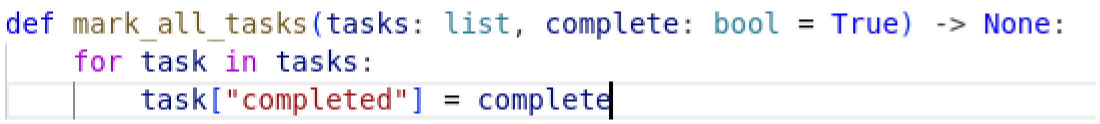
```python
def mark_all_tasks(tasks: list, complete: bool = True) -> None:
    pass
```

- Test failure demonstration



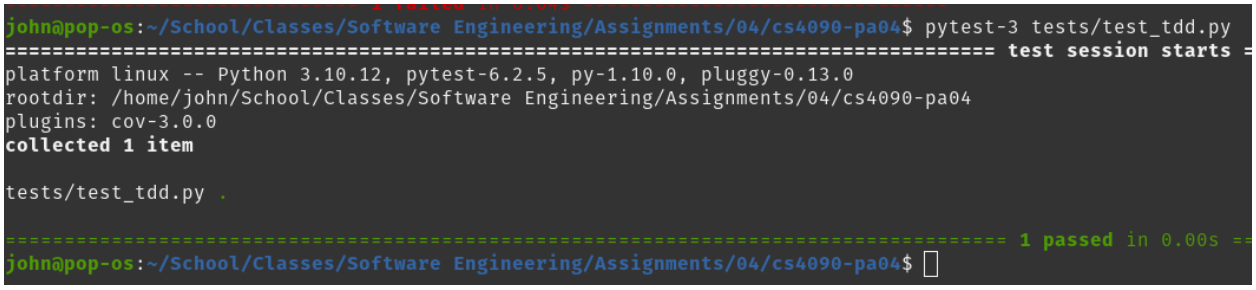
```
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ pytest-3 tests/test_tdd.py
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-6.2.5, py-1.10.0, pluggy-0.13.0
rootdir: /home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa04
plugins: cov-3.0.0
collected 1 item

tests/test_tdd.py F                                                      [100%]

=================================== FAILURES ===================================
______________________________ test_mark_all_tasks _____________________________

    def test_mark_all_tasks():
      tasks = [
        {"title": "Task 1", "completed": True},
        {"title": "Task 2", "completed": False},
        {"title": "Task 3", "completed": True},
      ]

      # test mark all true
      complete_tasks = copy.deepcopy(tasks)
      tsks.mark_all_tasks(complete_tasks, True)
>     assert all(task["completed"] for task in complete_tasks)
E     assert False
E      +  where False = all(<generator object test_mark_all_tasks.<locals>.<genexpr> at 0x713f44fc0120>)

tests/test_tdd.py:21: AssertionError
=========================== short test summary info ============================
FAILED tests/test_tdd.py::test_mark_all_tasks - assert False
============================== 1 failed in 0.04s ===============================
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ 
```

- Feature implementation

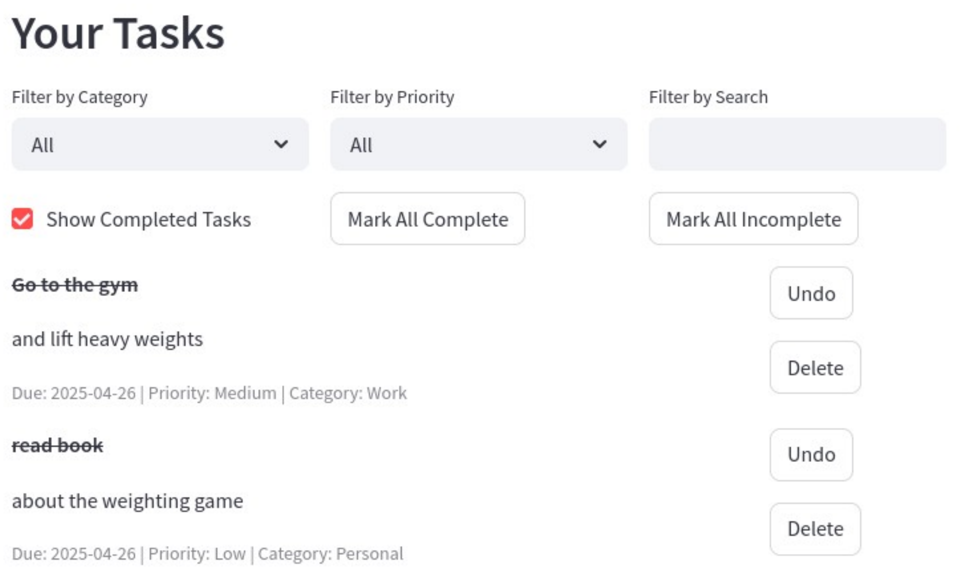


```
def mark_all_tasks(tasks: list, complete: bool = True) -> None:
    for task in tasks:
        task["completed"] = complete
```

- Test passing verification



```
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ pytest-3 tests/test_tdd.py
=================================================================================== test session starts =
platform linux -- Python 3.10.12, pytest-6.2.5, py-1.10.0, pluggy-0.13.0
rootdir: /home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa04
plugins: cov-3.0.0
collected 1 item

tests/test_tdd.py .

==================================================================================== 1 passed in 0.00s ==
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ 
```



Your Tasks

Filter by Category: All
Filter by Priority: All
Filter by Search:

[✓] Show Completed Tasks

~~Go to the gym~~

and lift heavy weights

Due: 2025-04-26 | Priority: Medium | Category: Work

~~read book~~

about the weighting game

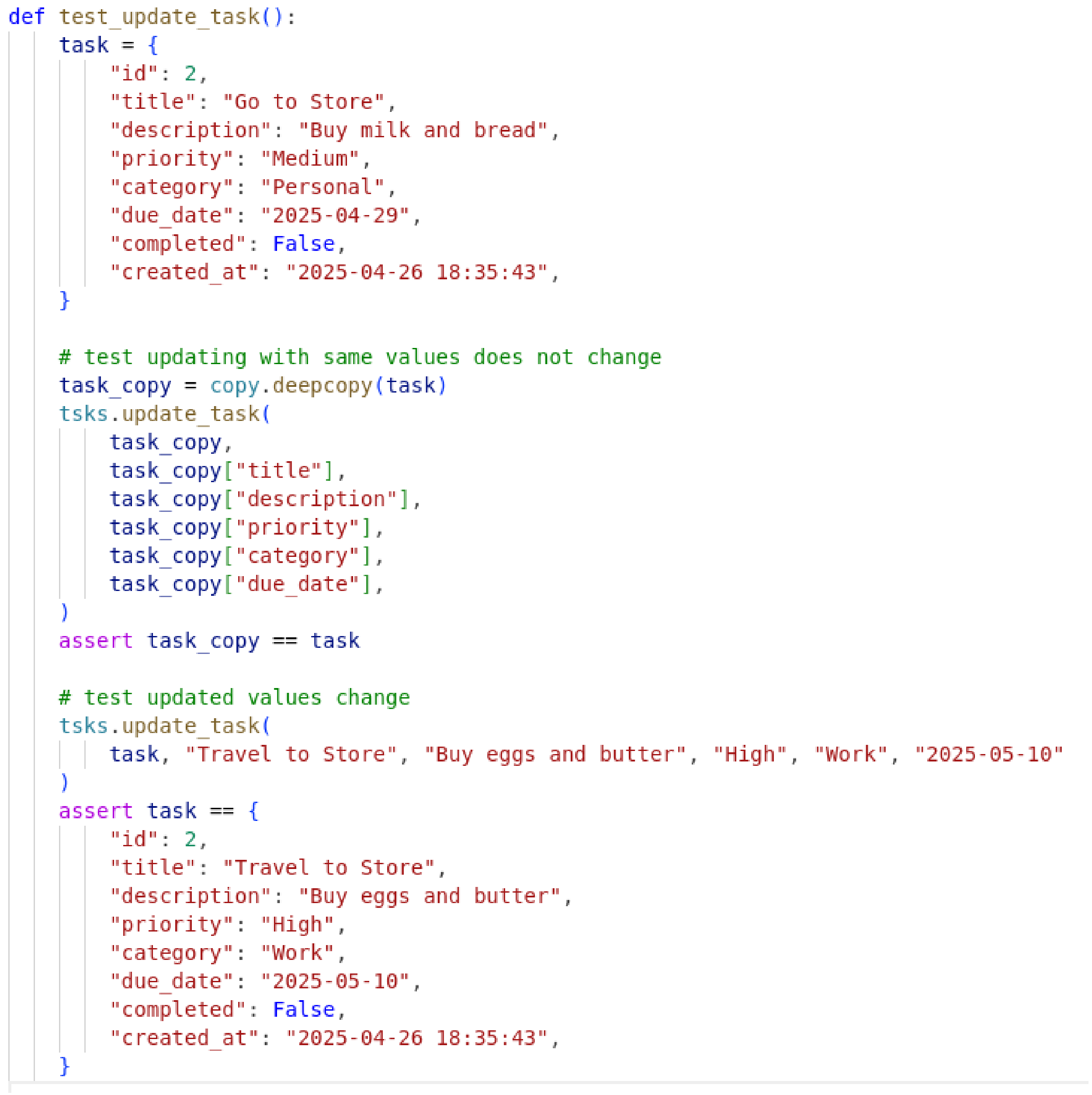
Due: 2025-04-26 | Priority: Low | Category: Personal

- Any refactoring performed

No refactoring was necessary

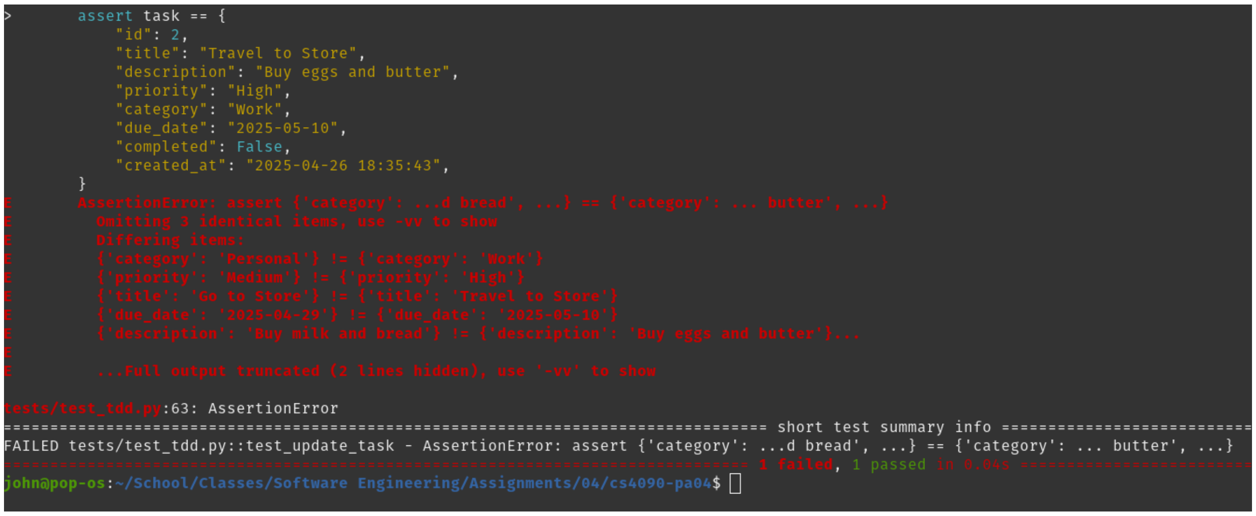
Feature 2: Task Editing with update_task()

- Initial test creation



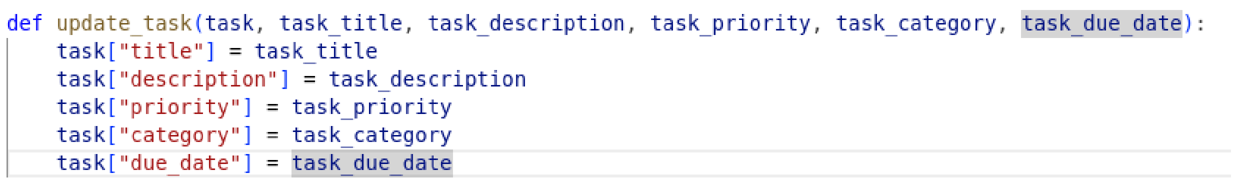
```python
def test_update_task():
    task = {
        "id": 2,
        "title": "Go to Store",
        "description": "Buy milk and bread",
        "priority": "Medium",
        "category": "Personal",
        "due_date": "2025-04-29",
        "completed": False,
        "created_at": "2025-04-26 18:35:43",
    }

    # test updating with same values does not change
    task_copy = copy.deepcopy(task)
    tsks.update_task(
        task_copy,
        task_copy["title"],
        task_copy["description"],
        task_copy["priority"],
        task_copy["category"],
        task_copy["due_date"],
    )
    assert task_copy == task

    # test updated values change
    tsks.update_task(
        task, "Travel to Store", "Buy eggs and butter", "High", "Work", "2025-05-10"
    )
    assert task == {
        "id": 2,
        "title": "Travel to Store",
        "description": "Buy eggs and butter",
        "priority": "High",
        "category": "Work",
        "due_date": "2025-05-10",
        "completed": False,
        "created_at": "2025-04-26 18:35:43",
    }
```

- Test failure demonstration

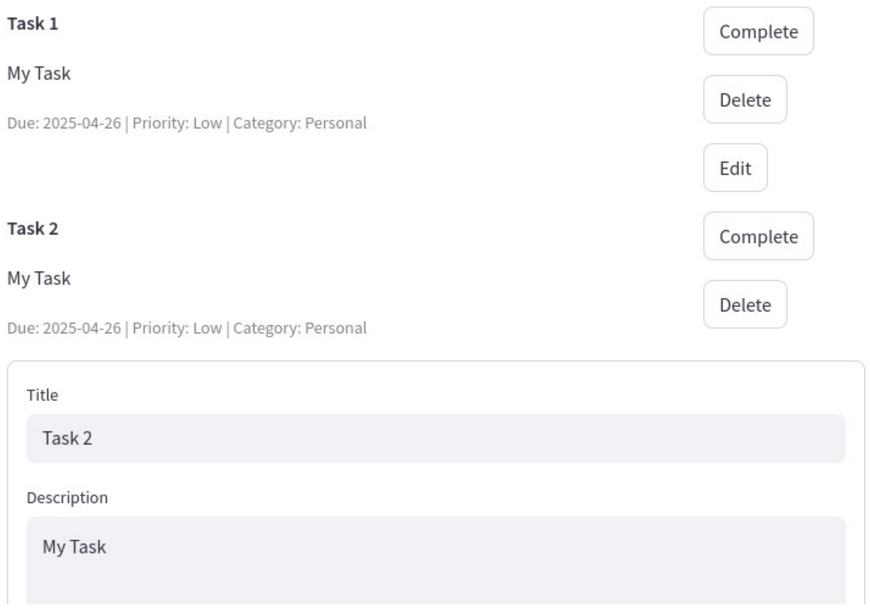


```
>       assert task == {
            "id": 2,
            "title": "Travel to Store",
            "description": "Buy eggs and butter",
            "priority": "High",
            "category": "Work",
            "due_date": "2025-05-10",
            "completed": False,
            "created_at": "2025-04-26 18:35:43",
        }
E       AssertionError: assert {'category': ...d bread', ...} == {'category': ... butter', ...}
E         Omitting 3 identical items, use -vv to show
E         Differing items:
E         {'category': 'Personal'} != {'category': 'Work'}
E         {'priority': 'Medium'} != {'priority': 'High'}
E         {'title': 'Go to Store'} != {'title': 'Travel to Store'}
E         {'due_date': '2025-04-29'} != {'due_date': '2025-05-10'}
E         {'description': 'Buy milk and bread'} != {'description': 'Buy eggs and butter'}...
E
E         ...Full output truncated (2 lines hidden), use '-vv' to show

tests/test_tdd.py:63: AssertionError
================================================================================ short test summary info ===========================
FAILED tests/test_tdd.py::test_update_task - AssertionError: assert {'category': ...d bread', ...} == {'category': ... butter', ...}
=============================================================================== 1 failed, 1 passed in 0.04s ========================
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$
```

- Feature implementation



```
def update_task(task, task_title, task_description, task_priority, task_category, task_due_date):
    task["title"] = task_title
    task["description"] = task_description
    task["priority"] = task_priority
    task["category"] = task_category
    task["due_date"] = task_due_date
```



Task 1

My Task

Due: 2025-04-26 | Priority: Low | Category: Personal

Task 2

My Task

Due: 2025-04-26 | Priority: Low | Category: Personal

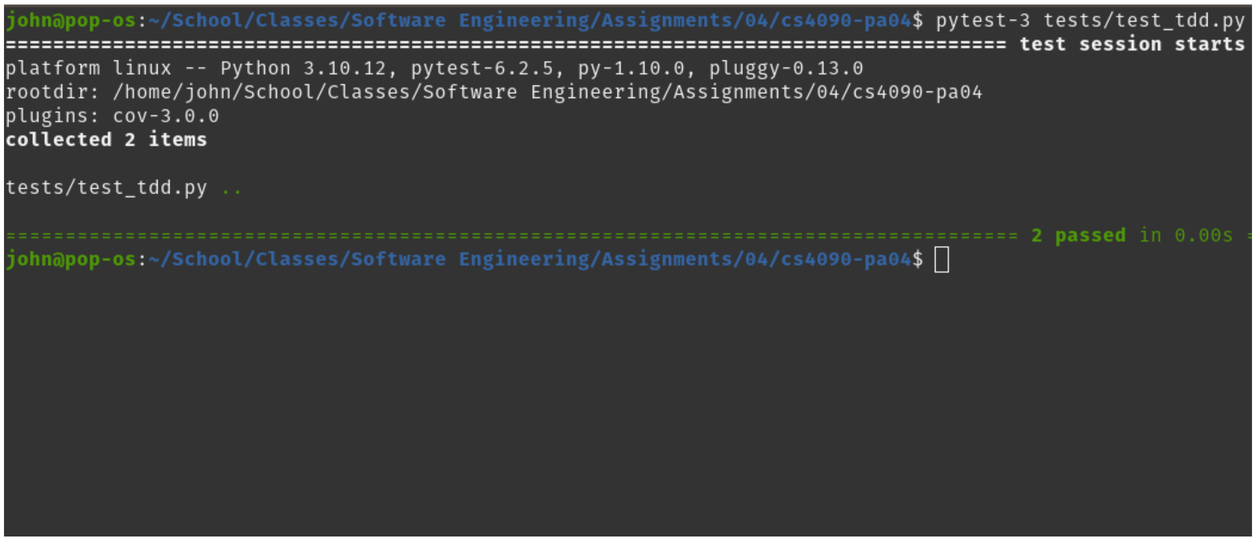
Title

Task 2

Description

My Task

- Test passing verification



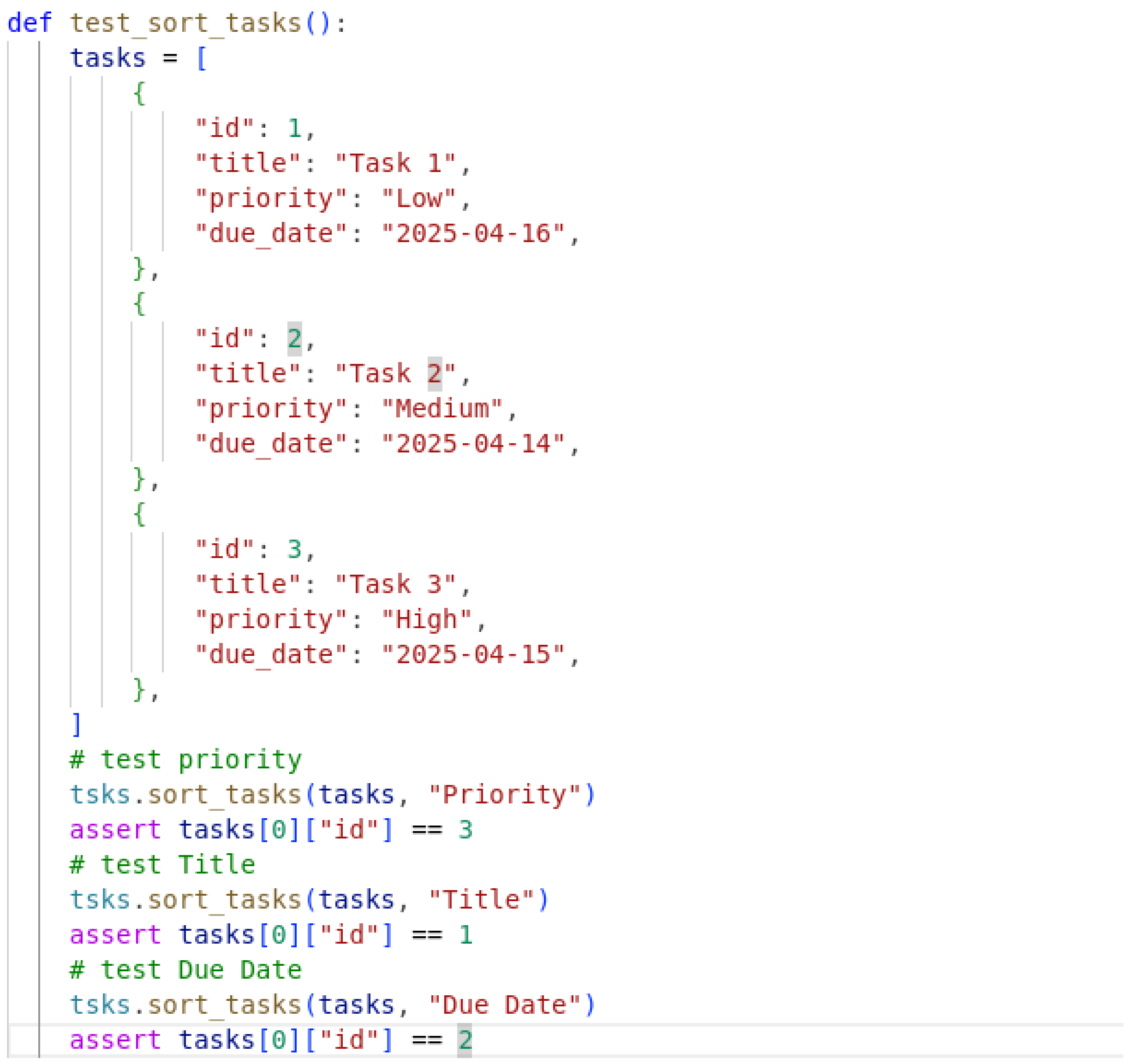
```
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ pytest-3 tests/test_tdd.py
=================================================================================== test session starts
platform linux -- Python 3.10.12, pytest-6.2.5, py-1.10.0, pluggy-0.13.0
rootdir: /home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa04
plugins: cov-3.0.0
collected 2 items

tests/test_tdd.py ..

==================================================================================== 2 passed in 0.00s =
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ 
```

- Any refactoring performed

No refactoring necessary

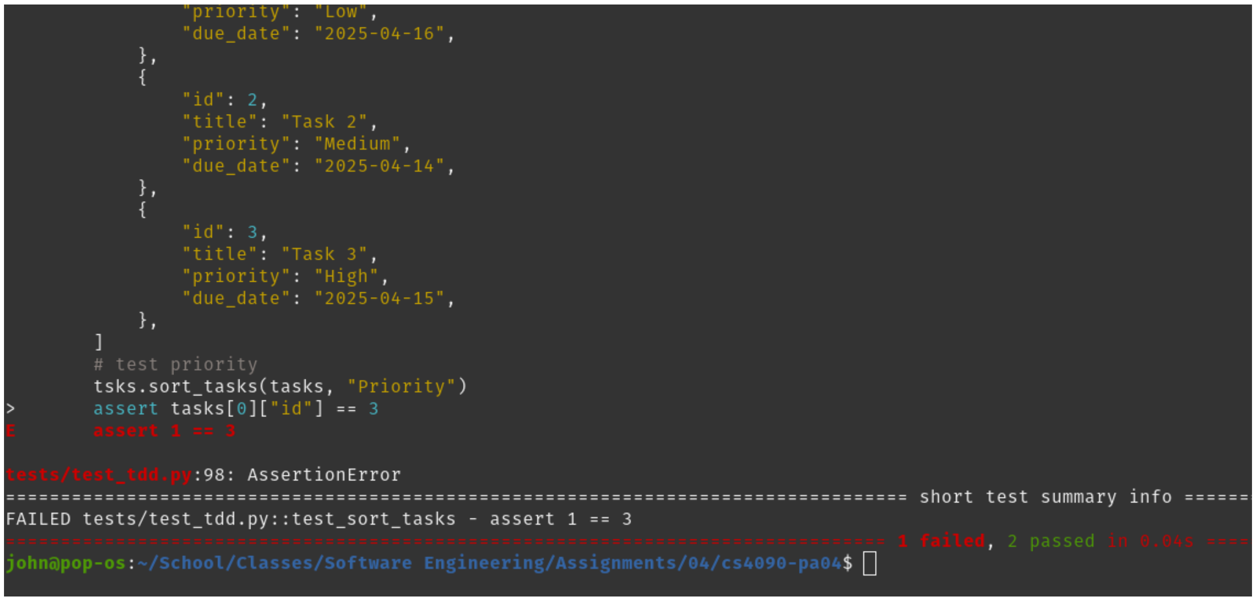
Feature 3: Sorted Task List

- Initial test creation



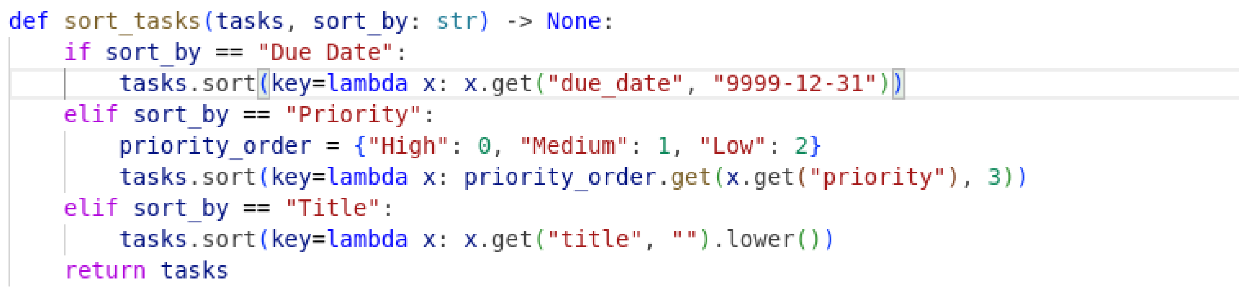
```python
def test_sort_tasks():
    tasks = [
        {
            "id": 1,
            "title": "Task 1",
            "priority": "Low",
            "due_date": "2025-04-16",
        },
        {
            "id": 2,
            "title": "Task 2",
            "priority": "Medium",
            "due_date": "2025-04-14",
        },
        {
            "id": 3,
            "title": "Task 3",
            "priority": "High",
            "due_date": "2025-04-15",
        },
    ]
    # test priority
    tsks.sort_tasks(tasks, "Priority")
    assert tasks[0]["id"] == 3
    # test Title
    tsks.sort_tasks(tasks, "Title")
    assert tasks[0]["id"] == 1
    # test Due Date
    tsks.sort_tasks(tasks, "Due Date")
    assert tasks[0]["id"] == 2
```

- Test failure demonstration

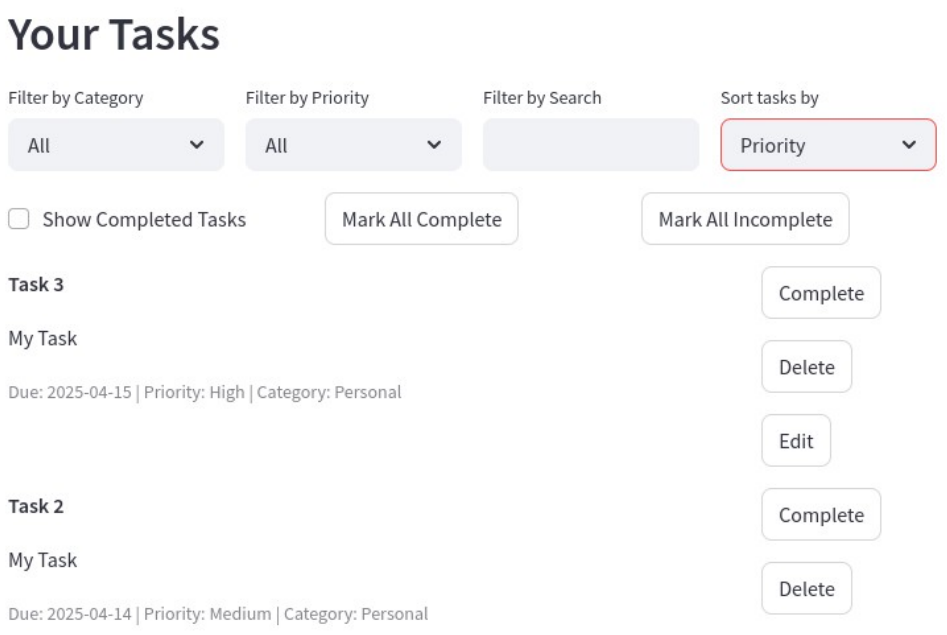


```
                "priority": "Low",
                "due_date": "2025-04-16",
            },
            {
                "id": 2,
                "title": "Task 2",
                "priority": "Medium",
                "due_date": "2025-04-14",
            },
            {
                "id": 3,
                "title": "Task 3",
                "priority": "High",
                "due_date": "2025-04-15",
            },
        ]
        # test priority
        tsks.sort_tasks(tasks, "Priority")
>       assert tasks[0]["id"] == 3
E       assert 1 == 3

tests/test_tdd.py:98: AssertionError
================================================================================== short test summary info ======
FAILED tests/test_tdd.py::test_sort_tasks - assert 1 == 3
================================================================================ 1 failed, 2 passed in 0.04s ====
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ 
```

- Feature implementation



```python
def sort_tasks(tasks, sort_by: str) -> None:
    if sort_by == "Due Date":
        tasks.sort(key=lambda x: x.get("due_date", "9999-12-31"))
    elif sort_by == "Priority":
        priority_order = {"High": 0, "Medium": 1, "Low": 2}
        tasks.sort(key=lambda x: priority_order.get(x.get("priority"), 3))
    elif sort_by == "Title":
        tasks.sort(key=lambda x: x.get("title", "").lower())
    return tasks
```



Your Tasks

Filter by Category: All
Filter by Priority: All
Filter by Search:
Sort tasks by: Priority

Task 3

My Task

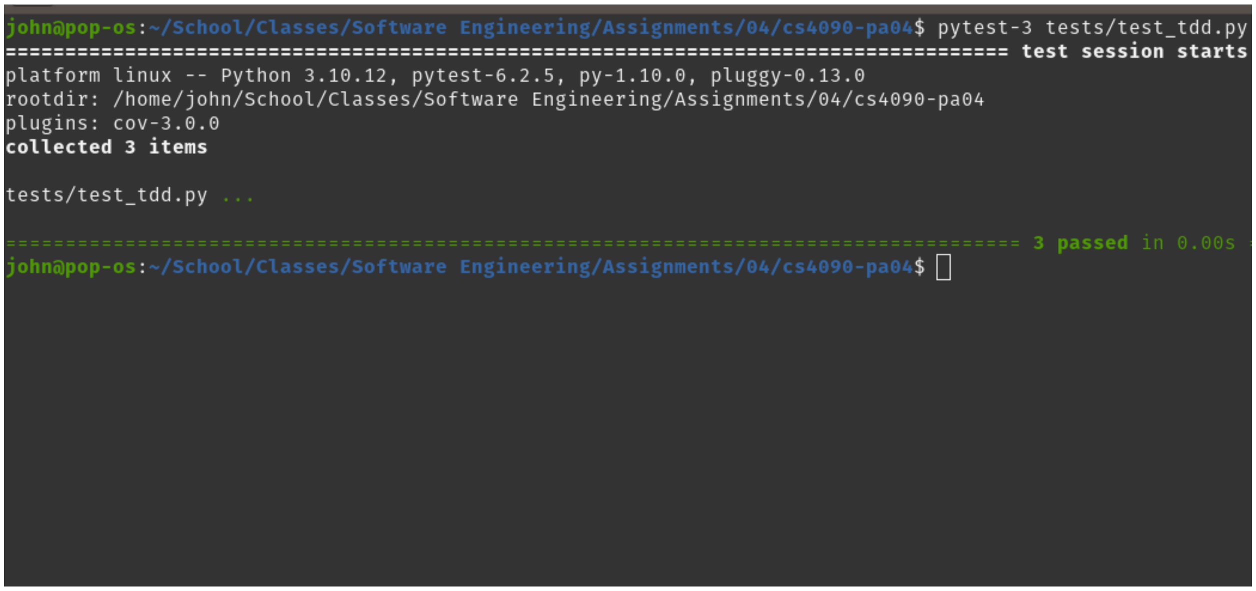
Due: 2025-04-15 | Priority: High | Category: Personal

Task 2

My Task

Due: 2025-04-14 | Priority: Medium | Category: Personal

- Test passing verification



```
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ pytest-3 tests/test_tdd.py
=================================================================================== test session starts
platform linux -- Python 3.10.12, pytest-6.2.5, py-1.10.0, pluggy-0.13.0
rootdir: /home/john/School/Classes/Software Engineering/Assignments/04/cs4090-pa04
plugins: cov-3.0.0
collected 3 items

tests/test_tdd.py ...

==================================================================================== 3 passed in 0.00s
john@pop-os:~/School/Classes/Software Engineering/Assignments/04/cs4090-pa04$ 
```

- Any refactoring performed

No refactoring was necessary

Part 5: Do Behavior-Driven Development (BDD)

For part 5 I implemented 5 BDD tests and added a button to the streamlit app to run them. The are located in tests/feature. The tasks.feature file contains the Gherkin syntax for my tests as required by the behave module, and the test_tasks.py file in the steps subdirectory contains the python code for running the tests. For my 5 tests, I chose to test the following functions:

- load_tasks()
- generate_unique_id()
- filter_tasks_by_priority()
- search_tasks
- sort_tasks

For each function, I thought of a Given-When-Then scenario with specific values and implemented the test accordingly

Part 6: Do Property-Based Testing (20 points)

For this part I used the hypothesis module to create 5 hypothesis tests for the functions in tasks.py. For each of these tests, I had to supply the @given decorator with a list of argument types to supply the test function with. For example, for test_generate_unqiue_id, in the @given decorator I had to specify a list of dictionaries with the value {“id”: <int>}. Then, when I run my tests with pytests, hypothesis runs my test functions many times with arbitrary values according to my given specification. This is a good testing strategy for finding edge cases and confirming correctness. When writing the test functions that take the arbitrary arguments, I just had to write them abstractly so they would be valid for any arguments that were passed. It was very similar to the parameterized unit testing.