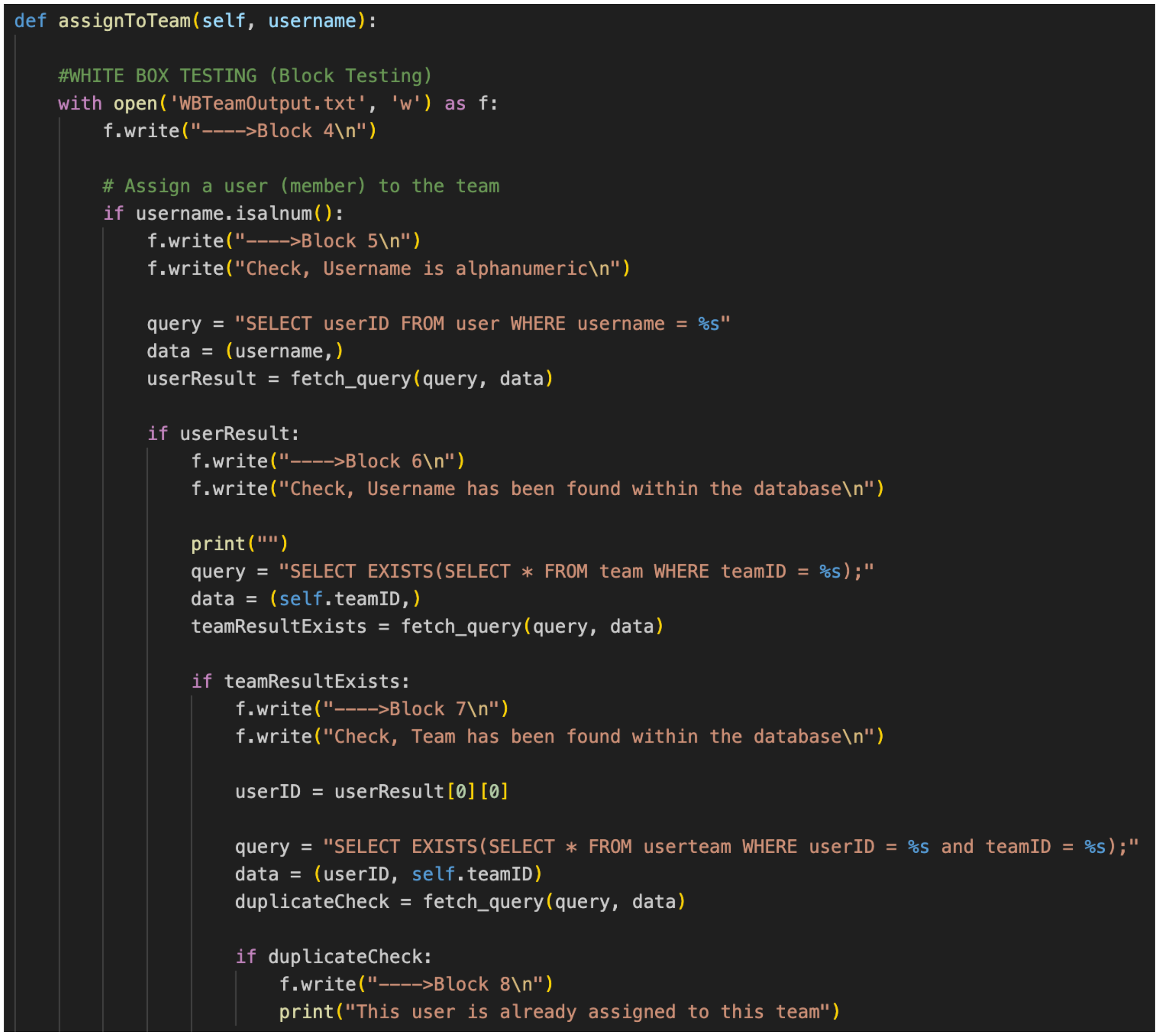
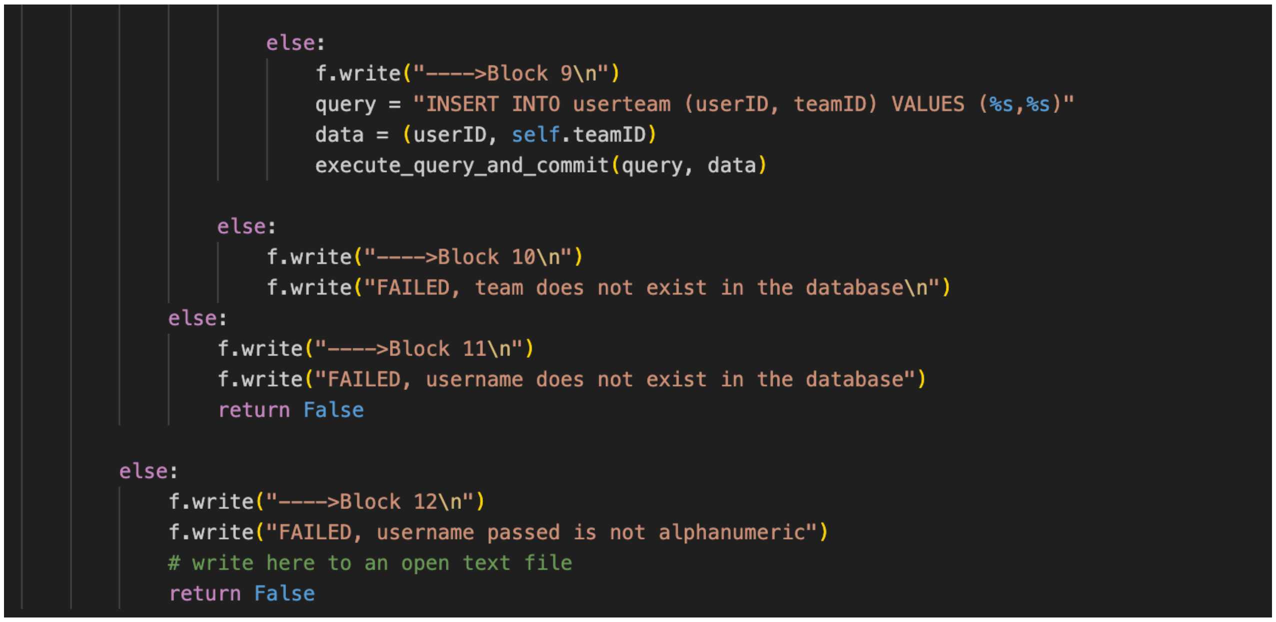
# Group 42 - White Box Testing

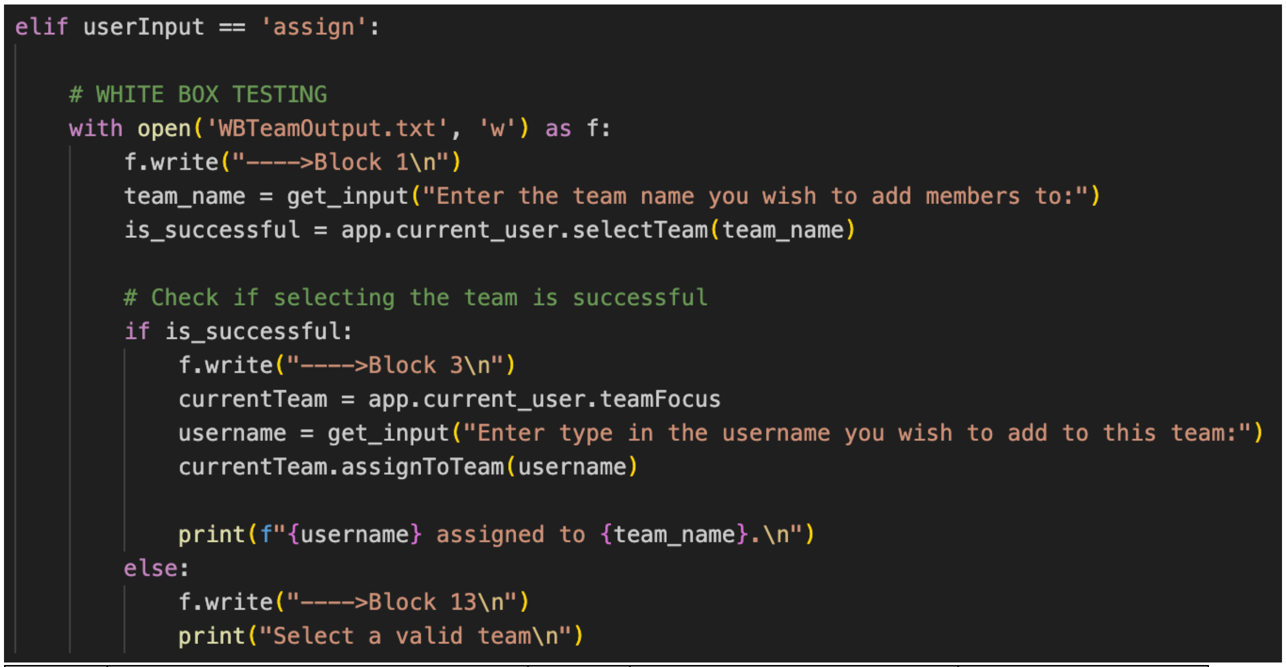
The first function that was white box tested was the assignToTeam function. This function is responsible for assigning a user to a team in the software. In order to white box test this, we used Block Testing. Here is the implementation below, comprised of most of it in the assignToTeam function while the rest in the main():

AssignToTeam:



```python
def assignToTeam(self, username):

    #WHITE BOX TESTING (Block Testing)
    with open('WBTeamOutput.txt', 'w') as f:
        f.write("---->Block 4\n")

        # Assign a user (member) to the team
        if username.isalnum():
            f.write("---->Block 5\n")
            f.write("Check, Username is alphanumeric\n")

            query = "SELECT userID FROM user WHERE username = %s"
            data = (username,)
            userResult = fetch_query(query, data)

            if userResult:
                f.write("---->Block 6\n")
                f.write("Check, Username has been found within the database\n")

                print("")
                query = "SELECT EXISTS(SELECT * FROM team WHERE teamID = %s);"
                data = (self.teamID,)
                teamResultExists = fetch_query(query, data)

                if teamResultExists:
                    f.write("---->Block 7\n")
                    f.write("Check, Team has been found within the database\n")

                    userID = userResult[0][0]

                    query = "SELECT EXISTS(SELECT * FROM userteam WHERE userID = %s and teamID = %s);"
                    data = (userID, self.teamID)
                    duplicateCheck = fetch_query(query, data)

                    if duplicateCheck:
                        f.write("---->Block 8\n")
                        print("This user is already assigned to this team")
```


```
                    else:
                        f.write("---->Block 9\n")
                        query = "INSERT INTO userteam (userID, teamID) VALUES (%s,%s)"
                        data = (userID, self.teamID)
                        execute_query_and_commit(query, data)

                else:
                    f.write("---->Block 10\n")
                    f.write("FAILED, team does not exist in the database\n")
            else:
                f.write("---->Block 11\n")
                f.write("FAILED, username does not exist in the database")
                return False

        else:
            f.write("---->Block 12\n")
            f.write("FAILED, username passed is not alphanumeric")
            # write here to an open text file
            return False
```
Main:


```
elif userInput == 'assign':

    # WHITE BOX TESTING
    with open('WBTeamOutput.txt', 'w') as f:
        f.write("---->Block 1\n")
        team_name = get_input("Enter the team name you wish to add members to:")
        is_successful = app.current_user.selectTeam(team_name)

        # Check if selecting the team is successful
        if is_successful:
            f.write("---->Block 3\n")
            currentTeam = app.current_user.teamFocus
            username = get_input("Enter type in the username you wish to add to this team:")
            currentTeam.assignToTeam(username)

            print(f"{username} assigned to {team_name}.\n")
        else:
            f.write("---->Block 13\n")
            print("Select a valid team\n")
```

| Block | Input | Test | Test Check | Output |
|---|---|---|---|---|
| 1 | assign | | Made it to the assign code in main | |
| 2 | *choosing a team | T1 | User has at least one team | |
| 3 | *choosing a team that exists for this user locally | T2 | Team focus selected | |
| 4 | | T3 | Made it to assign function | |

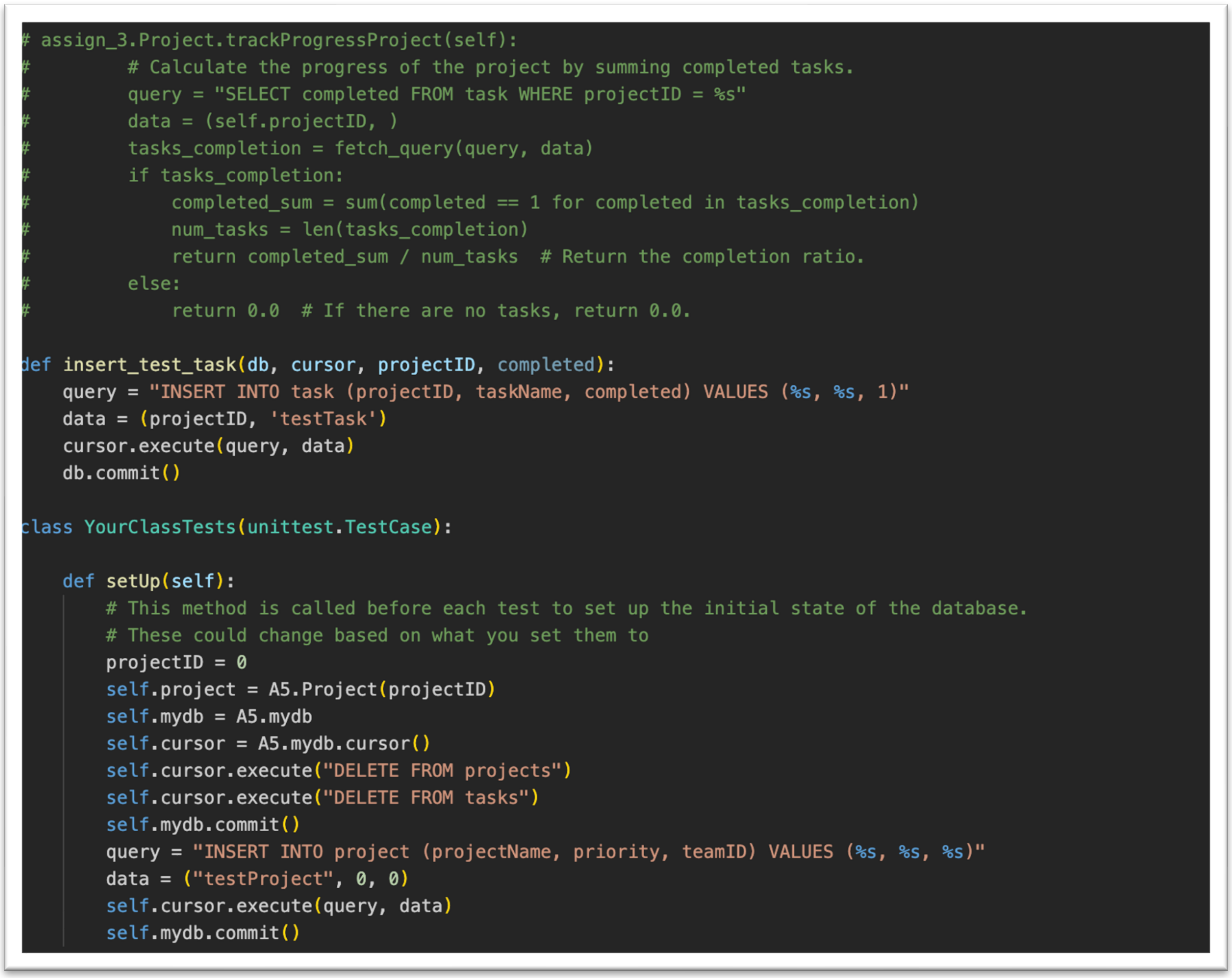
| | | | | |
|---|---|---|---|---|
| 5 | *Choosing an alphanumeric username | T4 | Member username contains only letters or numbers, no special characters | |
| 6 | *Member username given exists in database | T5 | Member username has been found in database | |
| 7 | *Team given exists in the database | T6 | | |
| 8 | *choose a member that is already part of that team | T7 | | member is already part of the team |
| 9 | *Choose a member that exists who is not already on a team that exists | | | Success |
| 10 | | | | team does not exist in the database |
| 11 | | | | username does not exist in database |
| 12 | | | | Username is not alphanumeric |
| 13 | | | | Select a valid team |

The following is the result of testing these cases given a valid username and team:

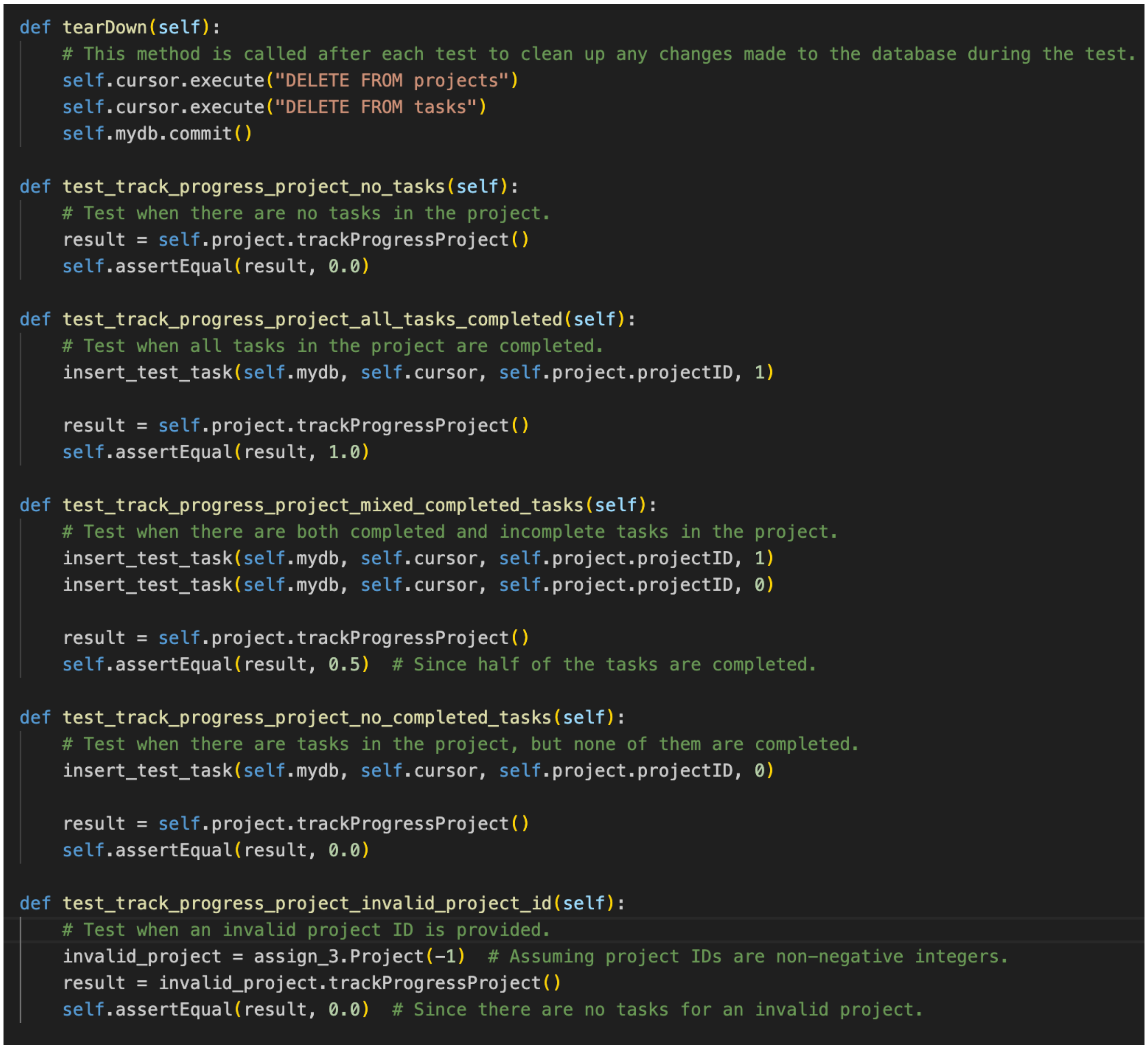
```
---->Block 1
---->Block 3
Check, Username is alphanumeric
---->Block 6
Check, Username has been found within the database
---->Block 7
Check, Team has been found within the database
---->Block 8
```

*The code writes the execution of the block testing into a separate txt file

The second function that was white box tested was the trackProgressProject function. This function is responsible for tracking the completion percentage of a project by dividing the completed tasks by the uncompleted tasks. In order to white box test this, we used Partition Testing insuring we understand exactly how the code works and making the tests to the exact specs of the code. Here is the implementation below:



```
# assign_3.Project.trackProgressProject(self):
#         # Calculate the progress of the project by summing completed tasks.
#         query = "SELECT completed FROM task WHERE projectID = %s"
#         data = (self.projectID, )
#         tasks_completion = fetch_query(query, data)
#         if tasks_completion:
#             completed_sum = sum(completed == 1 for completed in tasks_completion)
#             num_tasks = len(tasks_completion)
#             return completed_sum / num_tasks  # Return the completion ratio.
#         else:
#             return 0.0  # If there are no tasks, return 0.0.

def insert_test_task(db, cursor, projectID, completed):
    query = "INSERT INTO task (projectID, taskName, completed) VALUES (%s, %s, 1)"
    data = (projectID, 'testTask')
    cursor.execute(query, data)
    db.commit()

class YourClassTests(unittest.TestCase):

    def setUp(self):
        # This method is called before each test to set up the initial state of the database.
        # These could change based on what you set them to
        projectID = 0
        self.project = A5.Project(projectID)
        self.mydb = A5.mydb
        self.cursor = A5.mydb.cursor()
        self.cursor.execute("DELETE FROM projects")
        self.cursor.execute("DELETE FROM tasks")
        self.mydb.commit()
        query = "INSERT INTO project (projectName, priority, teamID) VALUES (%s, %s, %s)"
        data = ("testProject", 0, 0)
        self.cursor.execute(query, data)
        self.mydb.commit()
```



```python
def tearDown(self):
    # This method is called after each test to clean up any changes made to the database during the test.
    self.cursor.execute("DELETE FROM projects")
    self.cursor.execute("DELETE FROM tasks")
    self.mydb.commit()

def test_track_progress_project_no_tasks(self):
    # Test when there are no tasks in the project.
    result = self.project.trackProgressProject()
    self.assertEqual(result, 0.0)

def test_track_progress_project_all_tasks_completed(self):
    # Test when all tasks in the project are completed.
    insert_test_task(self.mydb, self.cursor, self.project.projectID, 1)

    result = self.project.trackProgressProject()
    self.assertEqual(result, 1.0)

def test_track_progress_project_mixed_completed_tasks(self):
    # Test when there are both completed and incomplete tasks in the project.
    insert_test_task(self.mydb, self.cursor, self.project.projectID, 1)
    insert_test_task(self.mydb, self.cursor, self.project.projectID, 0)

    result = self.project.trackProgressProject()
    self.assertEqual(result, 0.5)  # Since half of the tasks are completed.

def test_track_progress_project_no_completed_tasks(self):
    # Test when there are tasks in the project, but none of them are completed.
    insert_test_task(self.mydb, self.cursor, self.project.projectID, 0)

    result = self.project.trackProgressProject()
    self.assertEqual(result, 0.0)

def test_track_progress_project_invalid_project_id(self):
    # Test when an invalid project ID is provided.
    invalid_project = assign_3.Project(-1)  # Assuming project IDs are non-negative integers.
    result = invalid_project.trackProgressProject()
    self.assertEqual(result, 0.0)  # Since there are no tasks for an invalid project.
```

Here are the results of the testing:

Partition Test 1 – Track Project Progress with no tasks
This test function ensures that project progress is calculated properly, even with no tasks in a project.

| Task name / Completion<br>*all tasks under same project | Output (completion percent) |
|---|---|
| | |
| | 0% |

Partition Test 2 – Track Project Progress with all tasks completed
This test function ensures that project progress is calculated properly, with all tasks completed.

| Task name / Completion<br>*all tasks under same project | Output (completion percent) |
|---|---|
| Task1 = complete | 1 |
|  | 100% |

Partition Test 3 – Track Project Progress with mixed task completion

This test function ensures that project progress is calculated properly, with mixed task completion in a project.

| Task name / Completion<br>*all tasks under same project | Output (completion percent) |
|---|---|
| Task1 = complete | 1 |
| Task2 = incomplete | 0 |
|  | 50% |

Partition Test 4 – Track Project Progress with none of the tasks completed

This test function ensures that project progress is calculated properly, with all tasks incomplete.

| Task name / Completion<br>*all tasks under same project | Output (completion percent) |
|---|---|
| Task1 = incomplete | 0 |
|  | 0% |

Partition Test 5 – Track Project Progress with invalid project

This test function ensures that project progress is calculated properly, with invalid project selected.

| Task name / Completion<br>*all tasks under same project | Output (completion percent) |
|---|---|
| Project1 = does not exist |  |
|  | 0% |