ACCEPTANCE TESTING WORKSHOP

Ingeniería de Software II

Autor:
Juan Guadalupe

Juan Alejandro Guadalupe Rosas
[Dirección de correo electrónico]

Contenido

Repository

https://github.com/rexman10/Taller6-IS

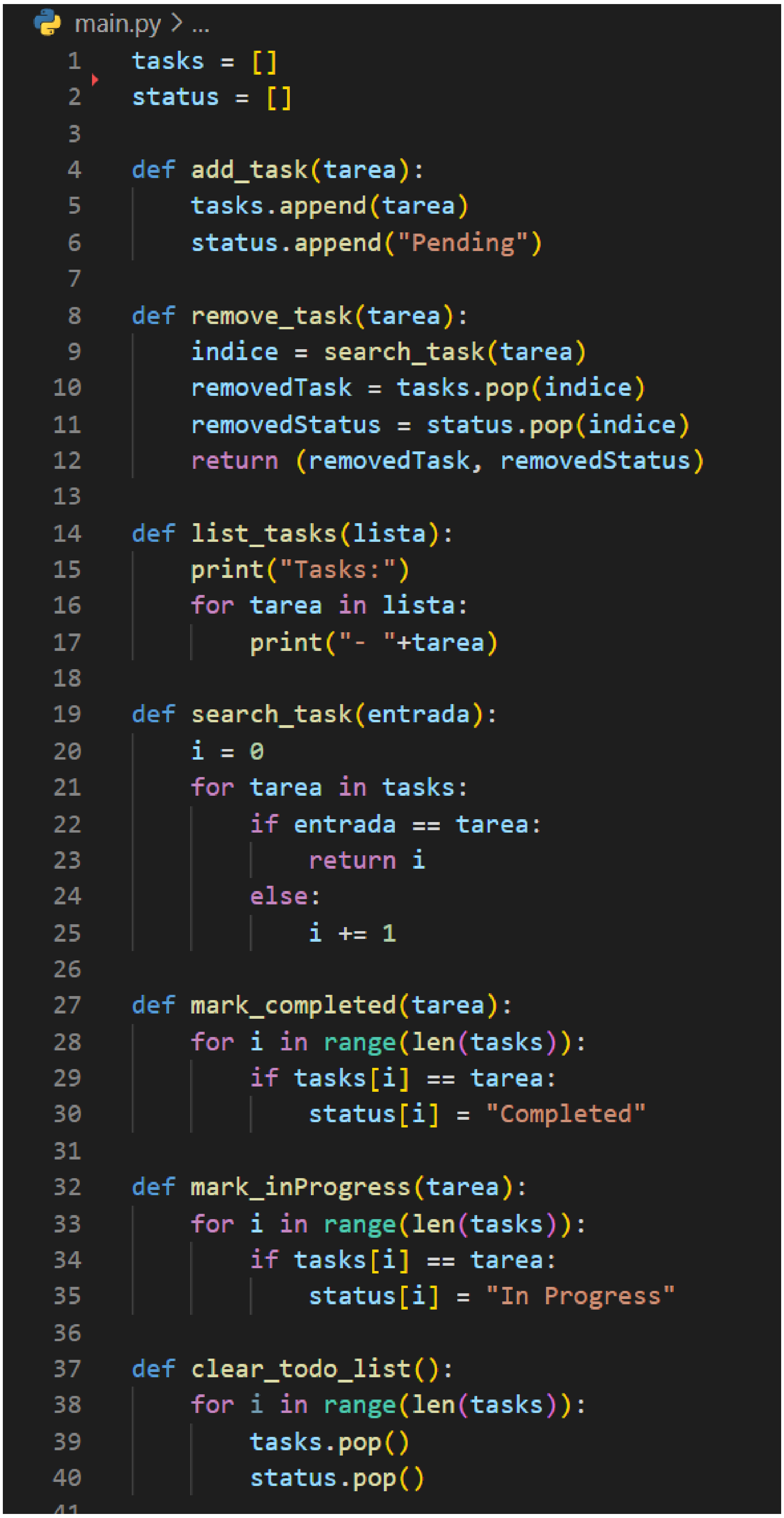
Introduction

The workshop report is all about this neat Python tool that helps handle to-do lists using the console. We taught people how to do stuff like add, delete, and update tasks with simple commands. Everybody got hands-on with Python scripting, and we made to-do list management really simple. The report tells the story of how people from different backgrounds learned to use the console and Python commands to make their task management super easy. It shares what we all learned together, how we used it in real situations, and the practical skills we gained in a straightforward and easy-to-understand way.

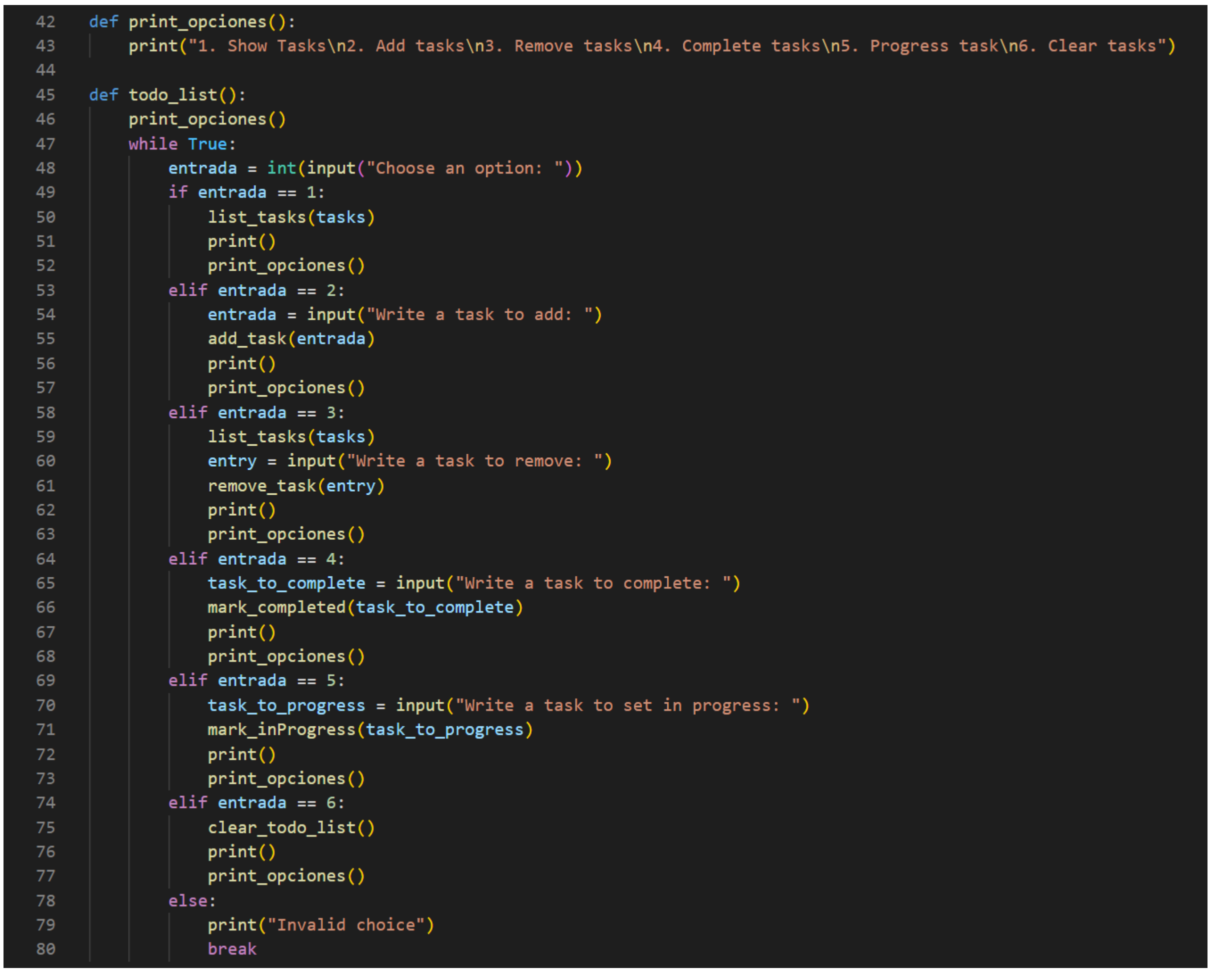
Developement

The development of this program consists on a series of functions that work together to complete the functionality of the to-do list.

To begin the process it's needed to write the core functions to be able to manipulate the tasks of the list.


main.py > ...
```python
tasks = []
status = []

def add_task(tarea):
    tasks.append(tarea)
    status.append("Pending")

def remove_task(tarea):
    indice = search_task(tarea)
    removedTask = tasks.pop(indice)
    removedStatus = status.pop(indice)
    return (removedTask, removedStatus)

def list_tasks(lista):
    print("Tasks:")
    for tarea in lista:
        print("- "+tarea)

def search_task(entrada):
    i = 0
    for tarea in tasks:
        if entrada == tarea:
            return i
        else:
            i += 1

def mark_completed(tarea):
    for i in range(len(tasks)):
        if tasks[i] == tarea:
            status[i] = "Completed"

def mark_inProgress(tarea):
    for i in range(len(tasks)):
        if tasks[i] == tarea:
            status[i] = "In Progress"

def clear_todo_list():
    for i in range(len(tasks)):
        tasks.pop()
        status.pop()
```

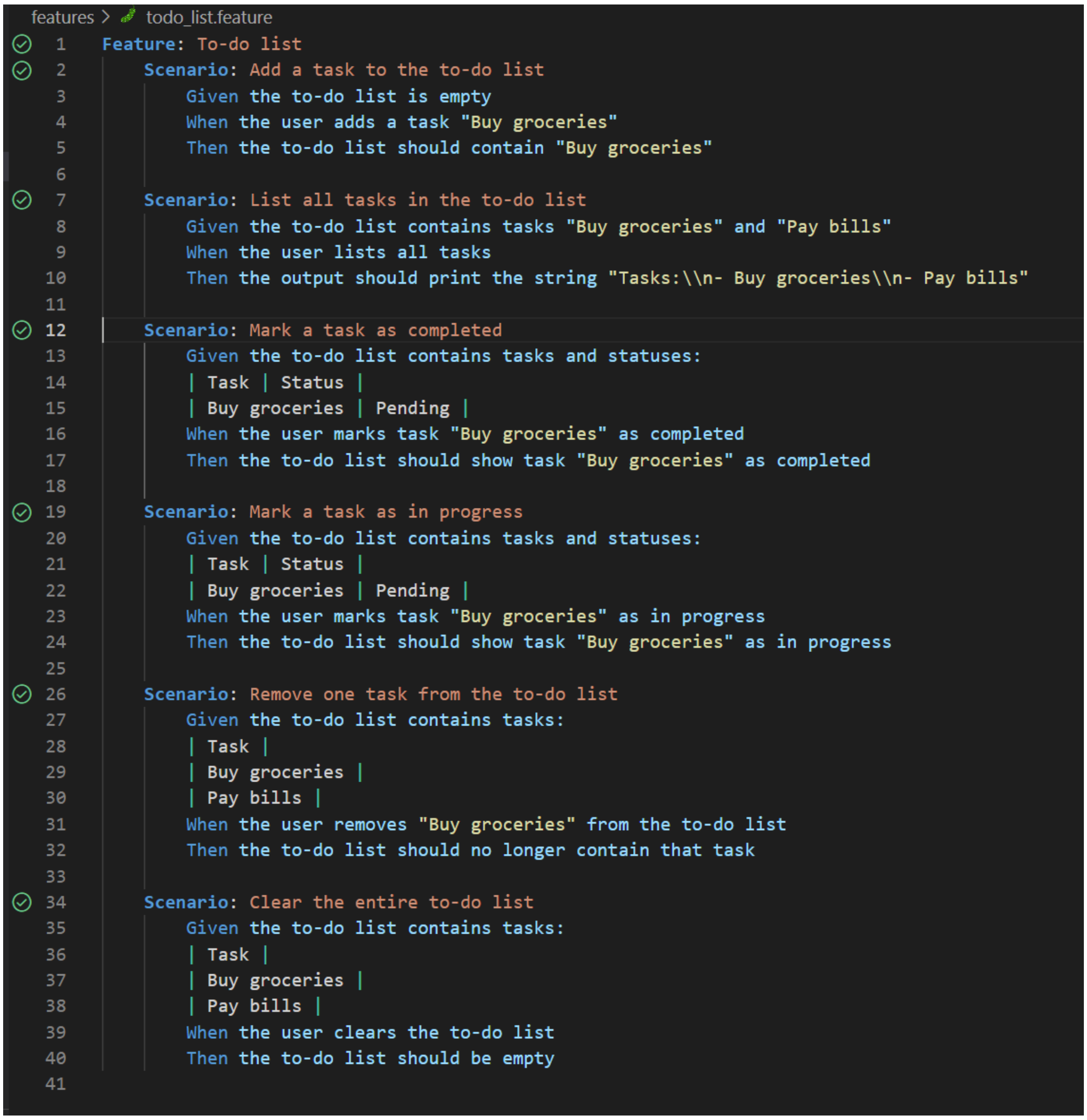
Once the core functions are done we need to manage the loop for the console program to works constantly until a certain input is entered by the user. This loop is achieved by implementing the following method.



```python
def print_opciones():
    print("1. Show Tasks\n2. Add tasks\n3. Remove tasks\n4. Complete tasks\n5. Progress task\n6. Clear tasks")

def todo_list():
    print_opciones()
    while True:
        entrada = int(input("Choose an option: "))
        if entrada == 1:
            list_tasks(tasks)
            print()
            print_opciones()
        elif entrada == 2:
            entrada = input("Write a task to add: ")
            add_task(entrada)
            print()
            print_opciones()
        elif entrada == 3:
            list_tasks(tasks)
            entry = input("Write a task to remove: ")
            remove_task(entry)
            print()
            print_opciones()
        elif entrada == 4:
            task_to_complete = input("Write a task to complete: ")
            mark_completed(task_to_complete)
            print()
            print_opciones()
        elif entrada == 5:
            task_to_progress = input("Write a task to set in progress: ")
            mark_inProgress(task_to_progress)
            print()
            print_opciones()
        elif entrada == 6:
            clear_todo_list()
            print()
            print_opciones()
        else:
            print("Invalid choice")
            break
```

The combination of functions implemented in the main.py file make possible to fulfill the requieremtns.

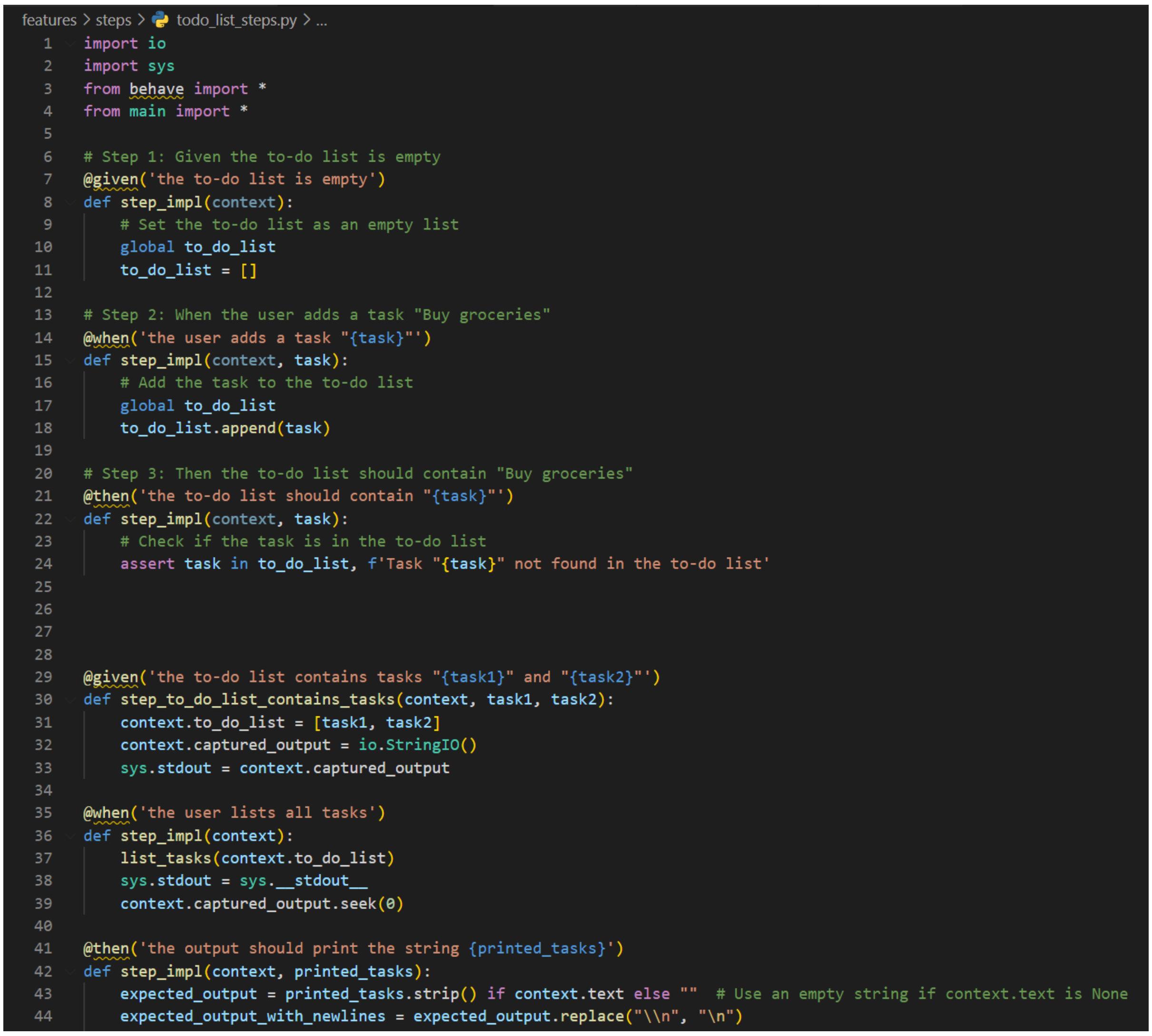
Now its required to create the acceptance test statements for every functionality in the program. The code to be implemented is required to be written according to a certain syntaxis, which is Gherkin, this includes every function as an acceptance test in the form of a "Feature".



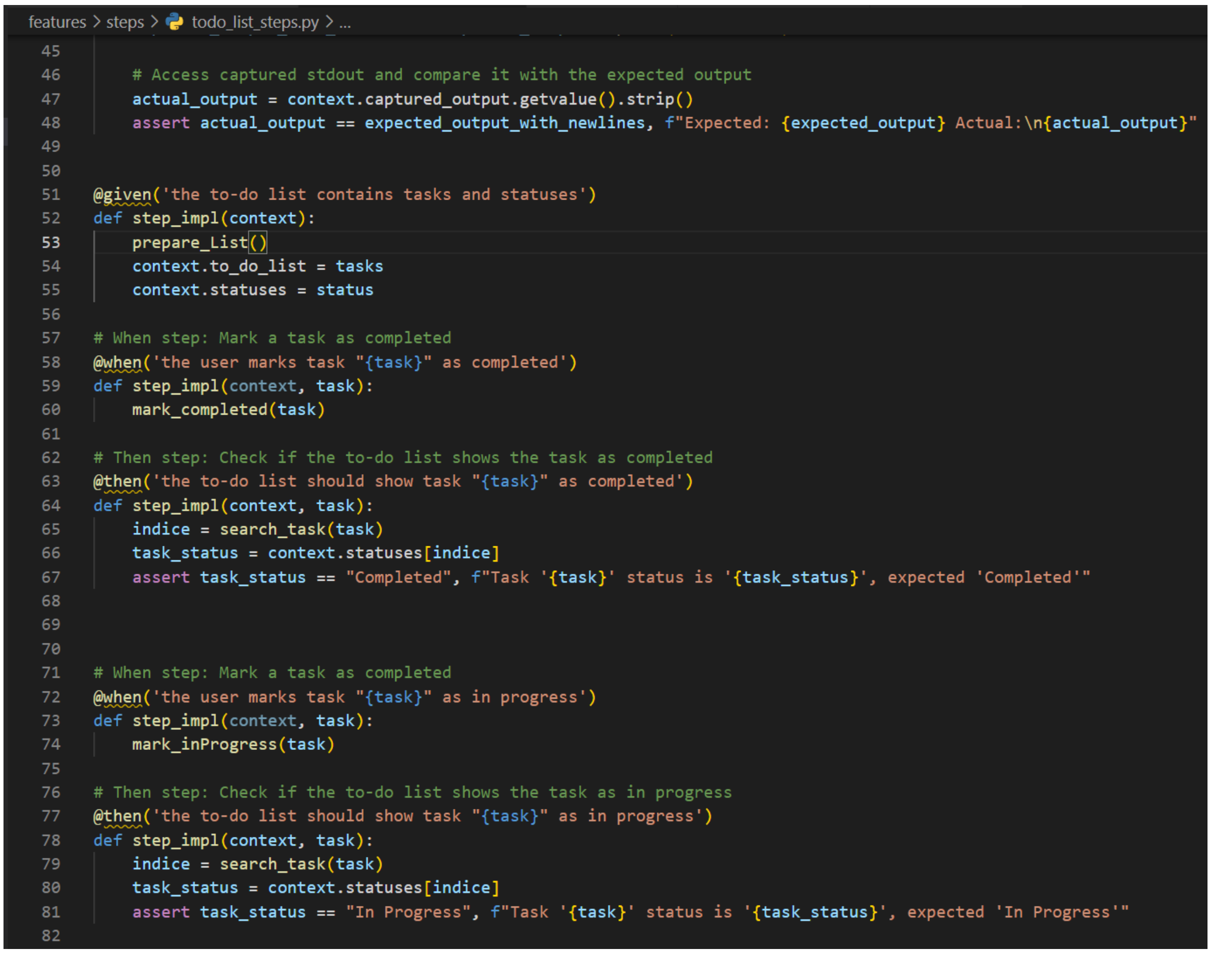
```
Feature: To-do list
    Scenario: Add a task to the to-do list
        Given the to-do list is empty
        When the user adds a task "Buy groceries"
        Then the to-do list should contain "Buy groceries"

    Scenario: List all tasks in the to-do list
        Given the to-do list contains tasks "Buy groceries" and "Pay bills"
        When the user lists all tasks
        Then the output should print the string "Tasks:\\n- Buy groceries\\n- Pay bills"

    Scenario: Mark a task as completed
        Given the to-do list contains tasks and statuses:
        | Task | Status |
        | Buy groceries | Pending |
        When the user marks task "Buy groceries" as completed
        Then the to-do list should show task "Buy groceries" as completed

    Scenario: Mark a task as in progress
        Given the to-do list contains tasks and statuses:
        | Task | Status |
        | Buy groceries | Pending |
        When the user marks task "Buy groceries" as in progress
        Then the to-do list should show task "Buy groceries" as in progress

    Scenario: Remove one task from the to-do list
        Given the to-do list contains tasks:
        | Task |
        | Buy groceries |
        | Pay bills |
        When the user removes "Buy groceries" from the to-do list
        Then the to-do list should no longer contain that task

    Scenario: Clear the entire to-do list
        Given the to-do list contains tasks:
        | Task |
        | Buy groceries |
        | Pay bills |
        When the user clears the to-do list
        Then the to-do list should be empty
```

Each of these scenarios establishes certain circumstances for each test to be validated.

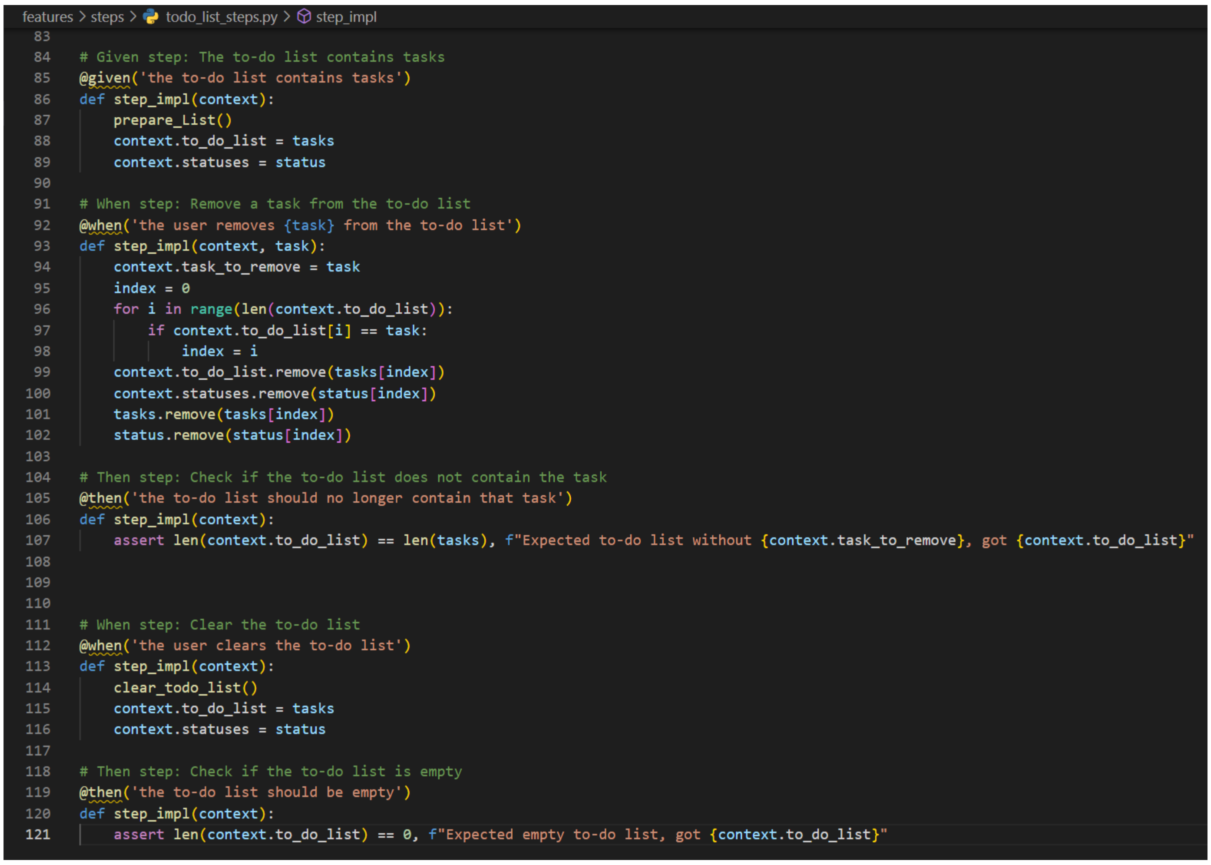
The next step for these tests to be validated is to write the assertion methods related to every scenario previously established, this is achieved by using the behave python library, it uses certain keywords to link the scenario to a Junit based test. The code written with behave is the following.


features > steps > todo_list_steps.py > ...

```python
import io
import sys
from behave import *
from main import *

# Step 1: Given the to-do list is empty
@given('the to-do list is empty')
def step_impl(context):
    # Set the to-do list as an empty list
    global to_do_list
    to_do_list = []

# Step 2: When the user adds a task "Buy groceries"
@when('the user adds a task "{task}"')
def step_impl(context, task):
    # Add the task to the to-do list
    global to_do_list
    to_do_list.append(task)

# Step 3: Then the to-do list should contain "Buy groceries"
@then('the to-do list should contain "{task}"')
def step_impl(context, task):
    # Check if the task is in the to-do list
    assert task in to_do_list, f'Task "{task}" not found in the to-do list'




@given('the to-do list contains tasks "{task1}" and "{task2}"')
def step_to_do_list_contains_tasks(context, task1, task2):
    context.to_do_list = [task1, task2]
    context.captured_output = io.StringIO()
    sys.stdout = context.captured_output

@when('the user lists all tasks')
def step_impl(context):
    list_tasks(context.to_do_list)
    sys.stdout = sys.__stdout__
    context.captured_output.seek(0)

@then('the output should print the string {printed_tasks}')
def step_impl(context, printed_tasks):
    expected_output = printed_tasks.strip() if context.text else ""  # Use an empty string if context.text is None
    expected_output_with_newlines = expected_output.replace("\\n", "\n")
```

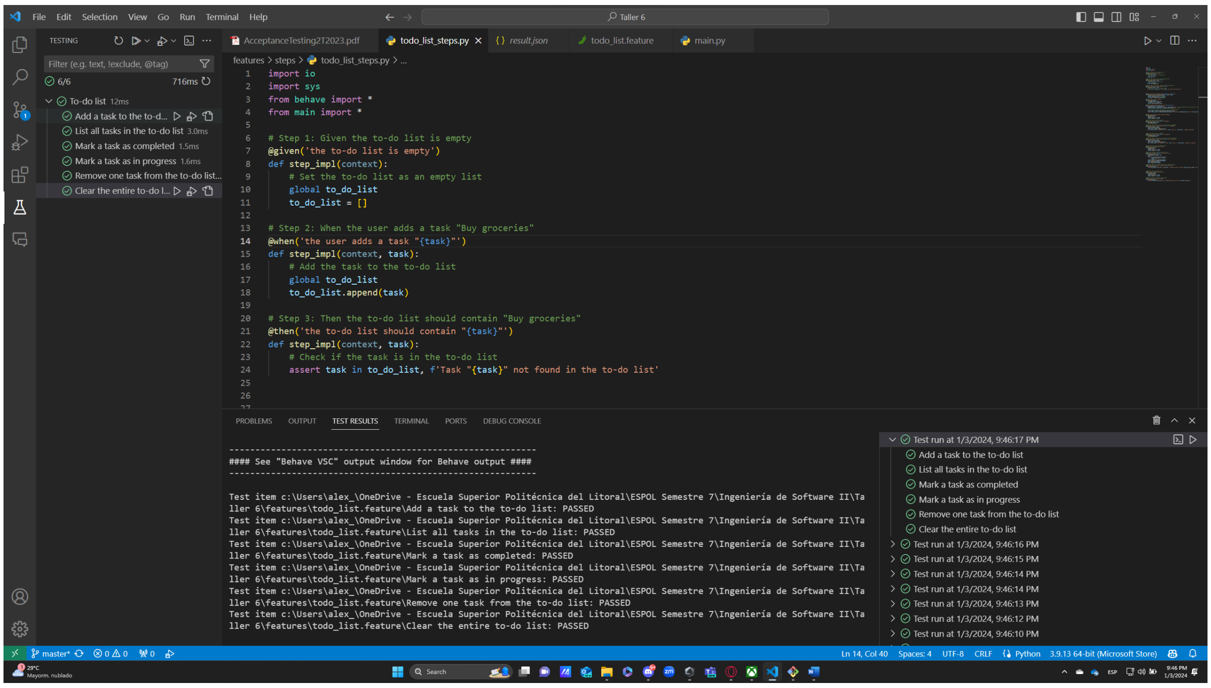


```python

    # Access captured stdout and compare it with the expected output
    actual_output = context.captured_output.getvalue().strip()
    assert actual_output == expected_output_with_newlines, f"Expected: {expected_output} Actual:\n{actual_output}"


@given('the to-do list contains tasks and statuses')
def step_impl(context):
    prepare_List()
    context.to_do_list = tasks
    context.statuses = status

# When step: Mark a task as completed
@when('the user marks task "{task}" as completed')
def step_impl(context, task):
    mark_completed(task)

# Then step: Check if the to-do list shows the task as completed
@then('the to-do list should show task "{task}" as completed')
def step_impl(context, task):
    indice = search_task(task)
    task_status = context.statuses[indice]
    assert task_status == "Completed", f"Task '{task}' status is '{task_status}', expected 'Completed'"



# When step: Mark a task as completed
@when('the user marks task "{task}" as in progress')
def step_impl(context, task):
    mark_inProgress(task)

# Then step: Check if the to-do list shows the task as in progress
@then('the to-do list should show task "{task}" as in progress')
def step_impl(context, task):
    indice = search_task(task)
    task_status = context.statuses[indice]
    assert task_status == "In Progress", f"Task '{task}' status is '{task_status}', expected 'In Progress'"

```



```python

# Given step: The to-do list contains tasks
@given('the to-do list contains tasks')
def step_impl(context):
    prepare_List()
    context.to_do_list = tasks
    context.statuses = status

# When step: Remove a task from the to-do list
@when('the user removes {task} from the to-do list')
def step_impl(context, task):
    context.task_to_remove = task
    index = 0
    for i in range(len(context.to_do_list)):
        if context.to_do_list[i] == task:
            index = i
    context.to_do_list.remove(tasks[index])
    context.statuses.remove(status[index])
    tasks.remove(tasks[index])
    status.remove(status[index])

# Then step: Check if the to-do list does not contain the task
@then('the to-do list should no longer contain that task')
def step_impl(context):
    assert len(context.to_do_list) == len(tasks), f"Expected to-do list without {context.task_to_remove}, got {context.to_do_list}"



# When step: Clear the to-do list
@when('the user clears the to-do list')
def step_impl(context):
    clear_todo_list()
    context.to_do_list = tasks
    context.statuses = status

# Then step: Check if the to-do list is empty
@then('the to-do list should be empty')
def step_impl(context):
    assert len(context.to_do_list) == 0, f"Expected empty to-do list, got {context.to_do_list}"
```

Once all the tests have been created the code must be tweaked until every one of them passes.



```python
import io
import sys
from behave import *
from main import *

# Step 1: Given the to-do list is empty
@given('the to-do list is empty')
def step_impl(context):
    # Set the to-do list as an empty list
    global to_do_list
    to_do_list = []

# Step 2: When the user adds a task "Buy groceries"
@when('the user adds a task "{task}"')
def step_impl(context, task):
    # Add the task to the to-do list
    global to_do_list
    to_do_list.append(task)

# Step 3: Then the to-do list should contain "Buy groceries"
@then('the to-do list should contain "{task}"')
def step_impl(context, task):
    # Check if the task is in the to-do list
    assert task in to_do_list, f'Task "{task}" not found in the to-do list'
```

```
-----------------------------------------------------------
#### See "Behave VSC" output window for Behave output ####
-----------------------------------------------------------

Test item c:\Users\alex_\OneDrive - Escuela Superior Politécnica del Litoral\ESPOL Semestre 7\Ingeniería de Software II\Ta
ller 6\features\todo_list.feature\Add a task to the to-do list: PASSED
Test item c:\Users\alex_\OneDrive - Escuela Superior Politécnica del Litoral\ESPOL Semestre 7\Ingeniería de Software II\Ta
ller 6\features\todo_list.feature\List all tasks in the to-do list: PASSED
Test item c:\Users\alex_\OneDrive - Escuela Superior Politécnica del Litoral\ESPOL Semestre 7\Ingeniería de Software II\Ta
ller 6\features\todo_list.feature\Mark a task as completed: PASSED
Test item c:\Users\alex_\OneDrive - Escuela Superior Politécnica del Litoral\ESPOL Semestre 7\Ingeniería de Software II\Ta
ller 6\features\todo_list.feature\Mark a task as in progress: PASSED
Test item c:\Users\alex_\OneDrive - Escuela Superior Politécnica del Litoral\ESPOL Semestre 7\Ingeniería de Software II\Ta
ller 6\features\todo_list.feature\Remove one task from the to-do list: PASSED
Test item c:\Users\alex_\OneDrive - Escuela Superior Politécnica del Litoral\ESPOL Semestre 7\Ingeniería de Software II\Ta
ller 6\features\todo_list.feature\Clear the entire to-do list: PASSED
```

TESTING
6/6
716ms
To-do list 12ms
Add a task to the to-d...
List all tasks in the to-do list 3.0ms
Mark a task as completed 1.5ms
Mark a task as in progress 1.6ms
Remove one task from the to-do list...
Clear the entire to-do l...
Test run at 1/3/2024, 9:46:17 PM
Add a task to the to-do list
List all tasks in the to-do list
Mark a task as completed
Mark a task as in progress
Remove one task from the to-do list
Clear the entire to-do list
Test run at 1/3/2024, 9:46:16 PM
Test run at 1/3/2024, 9:46:15 PM
Test run at 1/3/2024, 9:46:14 PM
Test run at 1/3/2024, 9:46:14 PM
Test run at 1/3/2024, 9:46:13 PM
Test run at 1/3/2024, 9:46:12 PM
Test run at 1/3/2024, 9:46:10 PM
Ln 14, Col 40
Python 3.9.13 64-bit (Microsoft Store)

Conclutions and recomendations

In summary, employing Gherkin language in tandem with Behave for testing our To-Do List Management program has emerged as a crucial strategy for validating its functionality and dependability. Gherkin's straightforward and human-readable syntax enabled us to articulate test scenarios in a manner easily comprehensible to both technical and non-technical stakeholders. Behave, serving as the connector between Gherkin and Python, facilitated smooth integration with our underlying codebase. Through a well-defined set of scenarios, we conducted thorough testing across various aspects of our program, encompassing tasks such as adding, completing, and handling diverse inputs. The combination of Gherkin and Behave not only delivered a robust suite of tests but also acted as living documentation, enhancing the transparency and manageability of our codebase. Rooted in behavior-driven development principles, this approach empowers our team to iterate confidently on the To-Do List Management program, ensuring both reliability and a user-centric experience.