Task 2:
Academic Planner Function

BANA 622
May 16, 2024

Emely Callejas, Ashley Cortez, Robert Pimentel, Angelica Verduzco

Table of Contents

Introduction

While navigating through academic institutions, students can find themselves struggling to keep track of the various assignments, projects, and deadlines that are assigned. This struggle often leads students to stress and could potentially cause them to miss deadlines. In order to address this issue, our team developed a tool designed to assist students in organizing their school schedule effectively. This tool is designed in a structured format to ensure accuracy and organization. Students can organize their school semester by inputting their classes, assignments, and deadlines. The student can reference back to this tool to ensure they are staying on track with their academics. A helpful feature this tool has is the flexibility in making changes and adjustments. Students can make changes to their schedule at any time and always have an updated version of their academic planner. This flexibility is crucial for the academic environment since changes can oftentimes occur unexpectedly. This tool is a helpful resource for students to assist in achieving academic success.

Methodology

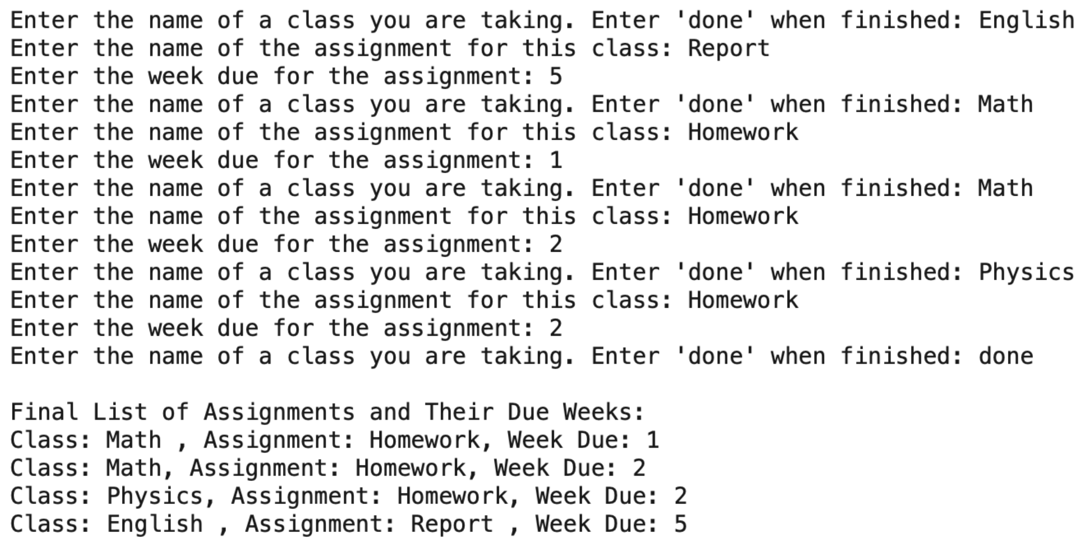
When creating our academic planner function we created the function and implemented it in the main iterative and recursive format. (Code snippets can be found in the appendix of this report.)

The iterative function itself was designed by first creating a global variable named ‘assignments’ which stores the details of each assignment. We then implemented a user input loop inside a collectAssignments function. This loop is designed to continue until a user inputs the term ‘done’ indicating that they do not have any other class names to enter. After each time the user inputs a class, the user is asked to input their assignments name as well as the week the assignment is due. We are then followed by appending a dictionary that contains the class name, assignments name, and week it is due within the ‘assignments’ list. Following this step, the main function is called that ultimately is calling the collectAssignments function. When all assignments are collected from the user, it starts sorting the ‘assignments’ list by the week that each assignment is due in ‘Week Due’. This is done by using the sort method in python and implementing a lambda function. After this is finalized, the function then prints the final list of the user's classes, assignments, sorted by the weeks they are due.

The other method we used was to create the function recursively. This function was created by implementing a list called ‘assignments’ followed by a recursive function named collectAssignmentsRec, which intakes the ‘assignments’ list as an argument. After this, the function requests for the user to input the name of the class they are taking. If the user enters ‘done’ the function will then return and act as the base case for this recursive method. When a user enters a class name, the function then will ask the user to enter the name of an assignment, and the week the assignment is due. Following this step, the function appends a dictionary which ultimately adds the new element to the assignment list. When the information for the assignment is collected, the function then uses recursion to call itself with the ‘collectAssignmentsRec(assignments)’ line to collect information for another assignment if needed. This is a repeated process until the user decides to input ‘done’. After the user inputs all of the assignments and enters ‘done’, the main function is called. The function then starts to sort the assignments in the list based on the week that each assignment is due by using the sort and lambda method. This is used to ensure that each assignment is then displayed in numerical order based on their due dates by week. Lastly, the main function prints the list of the assignments by the class name, assignment name, and week due using a loop.

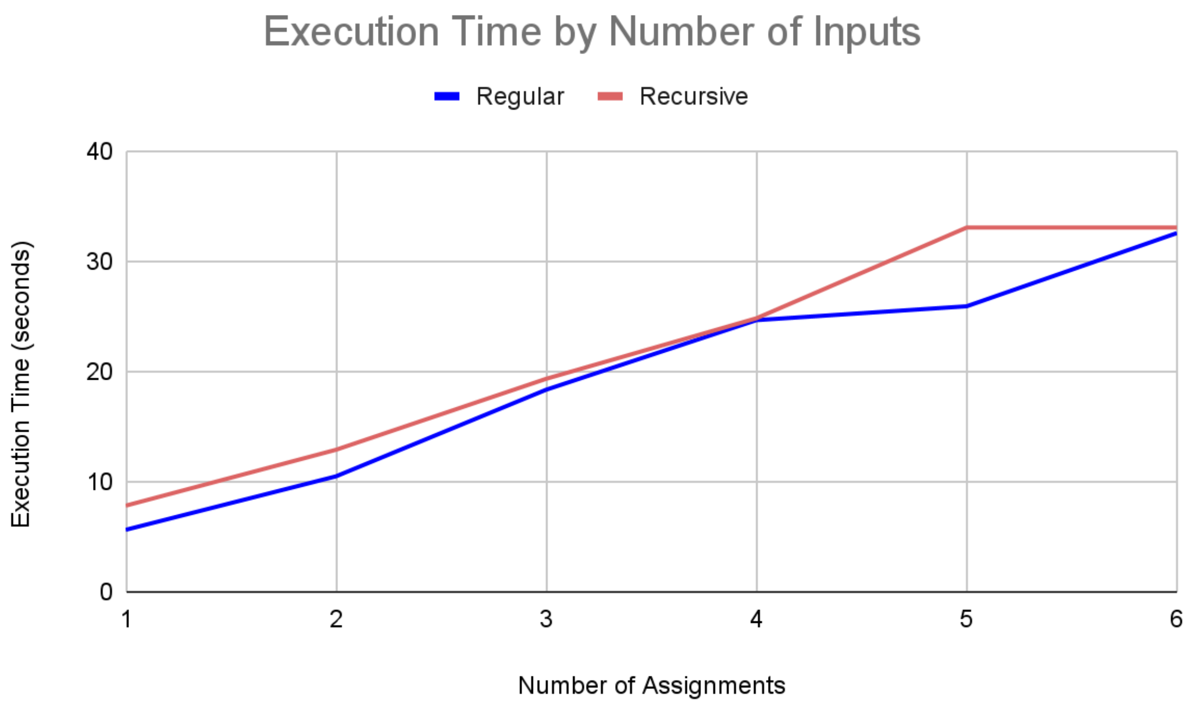
Results

For testing and validation purposes, we checked that each version of the function appropriately received user input for class name, assignment name, and week due. An essential portion of the code was that the output resulted in a final assignment list that was organized by the week due input. In order to test these results, we input assignments due out of order. Both the regular and recursive functions were able to successfully organize the assignment lists in the proper order, therefore validating that both functions were operating properly. Below is an example of one of our validation test cases:



```
Enter the name of a class you are taking. Enter 'done' when finished: English
Enter the name of the assignment for this class: Report
Enter the week due for the assignment: 5
Enter the name of a class you are taking. Enter 'done' when finished: Math
Enter the name of the assignment for this class: Homework
Enter the week due for the assignment: 1
Enter the name of a class you are taking. Enter 'done' when finished: Math
Enter the name of the assignment for this class: Homework
Enter the week due for the assignment: 2
Enter the name of a class you are taking. Enter 'done' when finished: Physics
Enter the name of the assignment for this class: Homework
Enter the week due for the assignment: 2
Enter the name of a class you are taking. Enter 'done' when finished: done

Final List of Assignments and Their Due Weeks:
Class: Math , Assignment: Homework, Week Due: 1
Class: Math, Assignment: Homework, Week Due: 2
Class: Physics, Assignment: Homework, Week Due: 2
Class: English , Assignment: Report , Week Due: 5
```

For our performance analysis, we conducted a series of tests for each version of the function. These test cases included recording the execution time by number of outputs for each function. We tested the time it would take to input 1-6 assignments for the regular version of the function and the recursive version of the function. Below is a visual representation of the results comparing the performance of the two versions:



As expected, the regular function typically provided the final output in slightly less time than the recursive function. As seen in the chart above, the difference in time for the two functions is relatively small. However, since this function primarily relies on input from the user, we found that the timing also relied heavily on how quickly the user input assignment information. For testing purposes we made sure to keep one name for class and assignment to keep variance at a minimum.

Discussion

As far as functionality, there wasn’t much of a difference between the accuracy or success of using a regular iterative function or a recursive function for this task.

If we were to expand this code to be more intuitive to take specific dates as inputs instead of just week numbers, there would be additional complexity added to the overall structure of the code. Taking this into consideration along with the task that was executed, we agreed that using a regular iterative function would be most beneficial in this scenario. Since there is a lot of user input involved with this function, there is a lot of room for error. We found that the format of the regular function was easier to read/understand, therefore making it easier to debug should there be an error. The recursive version of the function is slightly more complex to understand, specifically when it comes to understanding where/how the function stops calling on itself recursively to provide the final output. By nature, recursive functions are more prone to stack overflow, ultimately causing the program to crash if it continues to call on itself indefinitely after encountering an error.

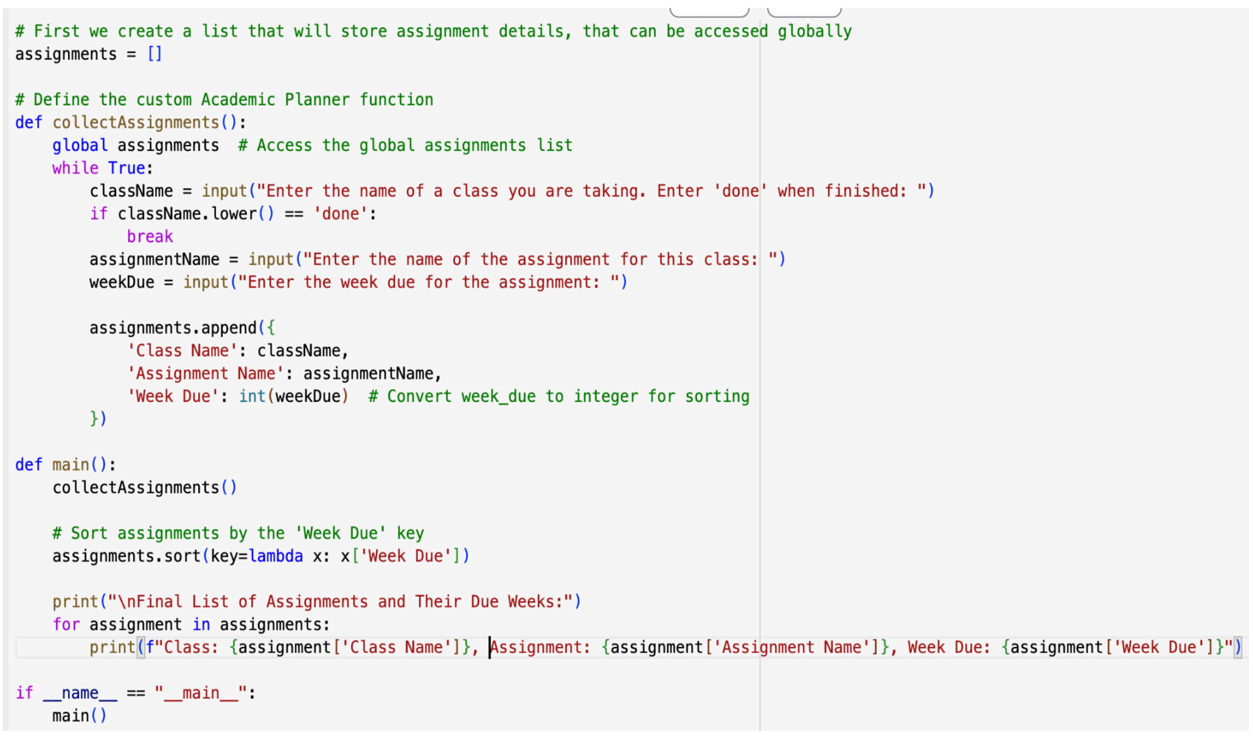
Conclusion

Through this exercise, we found that both regular and recursive functions were able to complete the task of collecting class names, assignment names and week due information to output a final assignment list sorted by week due.

For this particular function and its purpose, we ultimately recommend implementing the regular iterative function over the recursive function. We recommend using the iterative method because of its simplicity and efficiency.

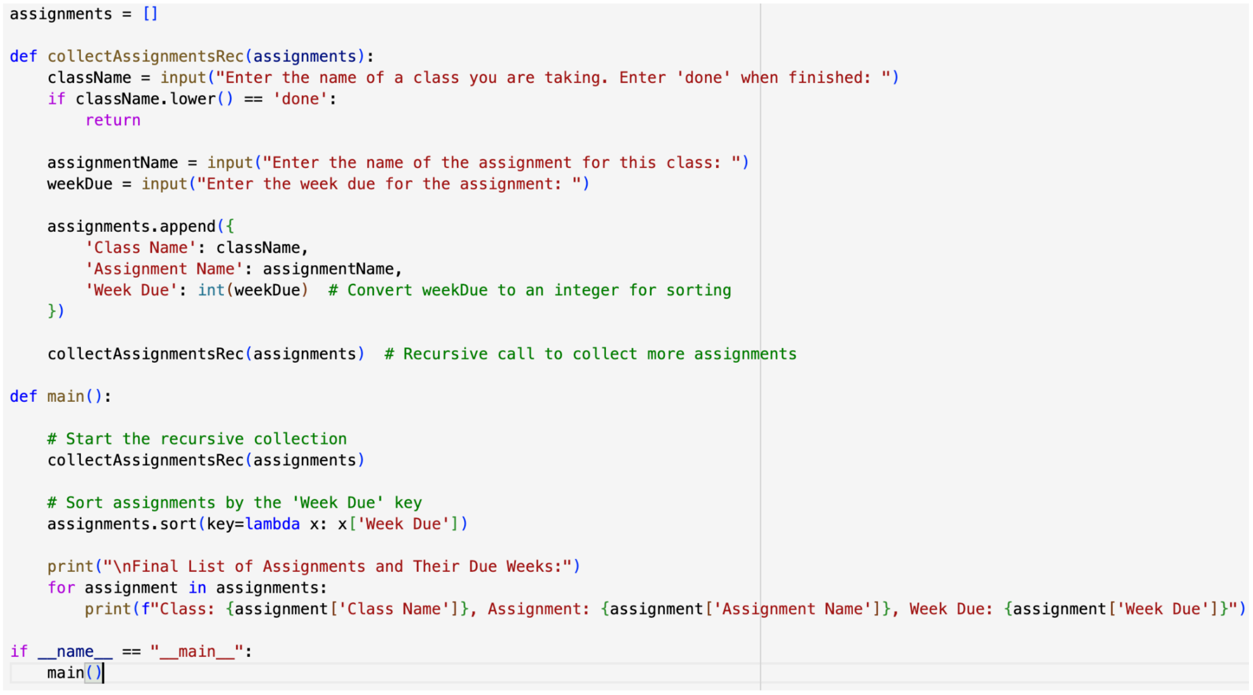
Appendix

Regular Function



```python
# First we create a list that will store assignment details, that can be accessed globally
assignments = []

# Define the custom Academic Planner function
def collectAssignments():
    global assignments  # Access the global assignments list
    while True:
        className = input("Enter the name of a class you are taking. Enter 'done' when finished: ")
        if className.lower() == 'done':
            break
        assignmentName = input("Enter the name of the assignment for this class: ")
        weekDue = input("Enter the week due for the assignment: ")

        assignments.append({
            'Class Name': className,
            'Assignment Name': assignmentName,
            'Week Due': int(weekDue)  # Convert week_due to integer for sorting
        })

def main():
    collectAssignments()

    # Sort assignments by the 'Week Due' key
    assignments.sort(key=lambda x: x['Week Due'])

    print("\nFinal List of Assignments and Their Due Weeks:")
    for assignment in assignments:
        print(f"Class: {assignment['Class Name']}, Assignment: {assignment['Assignment Name']}, Week Due: {assignment['Week Due']}")

if __name__ == "__main__":
    main()
```

Recursive Function:



```python
assignments = []

def collectAssignmentsRec(assignments):
    className = input("Enter the name of a class you are taking. Enter 'done' when finished: ")
    if className.lower() == 'done':
        return

    assignmentName = input("Enter the name of the assignment for this class: ")
    weekDue = input("Enter the week due for the assignment: ")

    assignments.append({
        'Class Name': className,
        'Assignment Name': assignmentName,
        'Week Due': int(weekDue)  # Convert weekDue to an integer for sorting
    })

    collectAssignmentsRec(assignments)  # Recursive call to collect more assignments

def main():

    # Start the recursive collection
    collectAssignmentsRec(assignments)

    # Sort assignments by the 'Week Due' key
    assignments.sort(key=lambda x: x['Week Due'])

    print("\nFinal List of Assignments and Their Due Weeks:")
    for assignment in assignments:
        print(f"Class: {assignment['Class Name']}, Assignment: {assignment['Assignment Name']}, Week Due: {assignment['Week Due']}")

if __name__ == "__main__":
    main()
```