

**Project :**

**Student Assignment Scheduler using SJF**

**Subject:**

**Operating System (Lab)**

**Group Members:**

**Sannia ilyas 105**

**Syeda maryam 111**

**Submitted To:**

**Sir Muneeb Saleem**

**Code:**

import datetime

import json

import os

from collections import defaultdict

# ✅ Define your custom file path in D drive

FILE\_PATH = r"/home/sannia-ilyas/Downloads/assignments.txt"

class Assignment:

def \_\_init\_\_(self, name, duration, deadline, status="Not Started", start\_time=None, end\_time=None):

self.name = name

self.duration = duration

self.deadline = datetime.datetime.strptime(deadline, "%Y-%m-%d")

self.status = status

self.start\_time = start\_time

self.end\_time = end\_time

def \_\_str\_\_(self):

return f"{self.name} | {self.duration}h | Deadline: {self.deadline.date()} | Status: {self.status}"

def to\_dict(self):

return {

"name": self.name,

"duration": self.duration,

"deadline": self.deadline.strftime("%Y-%m-%d"),

"status": self.status,

"start\_time": self.start\_time.strftime("%Y-%m-%d %H:%M:%S")if self.start\_time else None,

"end\_time": self.end\_time.strftime("%Y-%m-%d %H:%M:%S") if self.end\_time else None,

}

@staticmethod

def from\_dict(data):

return Assignment(

name=data["name"],

duration=data["duration"],

deadline=data["deadline"],

status=data.get("status", "Not Started"),

start\_time=datetime.datetime.strptime(data["start\_time"], "%Y-%m-%d %H:%M:%S") if data["start\_time"] else None,

end\_time=datetime.datetime.strptime(data["end\_time"], "%Y-%m-%d %H:%M:%S") if data["end\_time"] else None,

)

class Scheduler:

def \_\_init\_\_(self):

self.assignments = []

self.load\_assignments()

def save\_assignments(self):

os.makedirs(os.path.dirname(FILE\_PATH), exist\_ok=True)

with open(FILE\_PATH, "w") as f:

json.dump([a.to\_dict() for a in self.assignments], f, indent=4)

print(f"\n💾 Assignments saved to: {FILE\_PATH}")

def load\_assignments(self):

if os.path.exists(FILE\_PATH):

with open(FILE\_PATH, "r") as f:

data = json.load(f)

self.assignments = [Assignment.from\_dict(d) for d in data]

print(f"\n📂 Loaded existing assignments from: {FILE\_PATH}")

else:

print(f"\n📁 No existing file found at: {FILE\_PATH} (starting fresh)")

def add\_assignment(self, name, duration, deadline):

assignment = Assignment(name, duration, deadline)

self.assignments.append(assignment)

self.save\_assignments()

def sort\_by\_sjf(self):

self.assignments.sort(key=lambda x: x.duration)

def show\_assignments(self):

self.sort\_by\_sjf()

print("\n--- 📋 SJF Assignment List ---")

for idx, a in enumerate(self.assignments, start=1):

print(f"{idx}. {a}")

def mark\_complete(self, index):

if 0 <= index < len(self.assignments):

assignment = self.assignments[index]

assignment.status = "Completed"

assignment.end\_time = datetime.datetime.now()

self.save\_assignments()

print(f"✅ '{assignment.name}' marked as completed!")

else:

print("❌ Invalid assignment index.")

def report(self):

print("\n--- 📈 Completion Report ---")

for a in self.assignments:

if a.status == "Completed":

actual\_time = (a.end\_time - a.deadline).days

print(f"{a.name} | Completed On: {a.end\_time.date()} | Deadline: {a.deadline.date()} | Days Difference: {actual\_time}")

print("Total Completed:", sum(1 for a in self.assignments if a.status == "Completed"))

def show\_weekly\_calendar(self):

print("\n📅 Weekly Assignment Calendar View:")

week\_map = defaultdict(list)

for a in self.assignments:

week\_num = a.deadline.isocalendar()[1]

week\_map[week\_num].append(a)

for week in sorted(week\_map.keys()):

print(f"\n📆 Week {week}")

for a in sorted(week\_map[week], key=lambda x: x.deadline):

print(f" - {a.name} | Due: {a.deadline.strftime('%A, %Y-%m-%d')} | Status: {a.status}")

def show\_reminders(self):

today = datetime.datetime.now().date()

print("\n⏰ Reminders:")

found = False

for a in self.assignments:

if a.status != "Completed":

days\_left = (a.deadline.date() - today).days

if days\_left == 0:

print(f"⚠️ TODAY: '{a.name}' is due today!")

found = True

elif 0 < days\_left <= 2:

print(f"📌 UPCOMING: '{a.name}' is due in {days\_left} day(s).")

found = True

if not found:

print("🎉 No urgent deadlines today or in next 2 days.")

def main():

scheduler = Scheduler()

while True:

print("\n📚 Student Assignment Scheduler (SJF)")

print("1. Add Assignment(s)")

print("2. Show Assignments (SJF Order)")

print("3. Mark Assignment as Completed")

print("4. Show Completion Report")

print("5. Show Weekly Calendar View")

print("6. Show Deadline Reminders")

print("7. Exit")

choice = input("Select an option: ")

if choice == '1':

try:

count = int(input("How many assignments do you want to add? "))

for i in range(count):

print(f"\nAssignment {i + 1}:")

name = input(" Name: ")

duration = float(input(" Estimated Duration (in hours): "))

deadline = input(" Deadline (YYYY-MM-DD): ")

scheduler.add\_assignment(name, duration, deadline)

print(f"\n✅ {count} assignment(s) added successfully!")

except ValueError:

print("❌ Invalid input. Please enter a valid number.")

elif choice == '2':

scheduler.show\_assignments()

elif choice == '3':

scheduler.show\_assignments()

try:

idx = int(input("Enter assignment number to mark as completed: ")) - 1

scheduler.mark\_complete(idx)

except ValueError:

print("❌ Please enter a valid number.")

elif choice == '4':

scheduler.report()

elif choice == '5':

scheduler.show\_weekly\_calendar()

elif choice == '6':

scheduler.show\_reminders()

elif choice == '7':

print("👋 Exiting. Stay on top of your deadlines!")

break

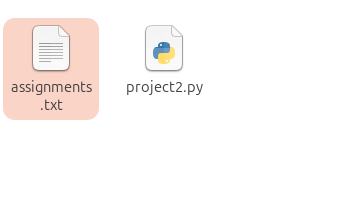
else:

print("❗ Invalid option, try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

* **File pictures:**

****

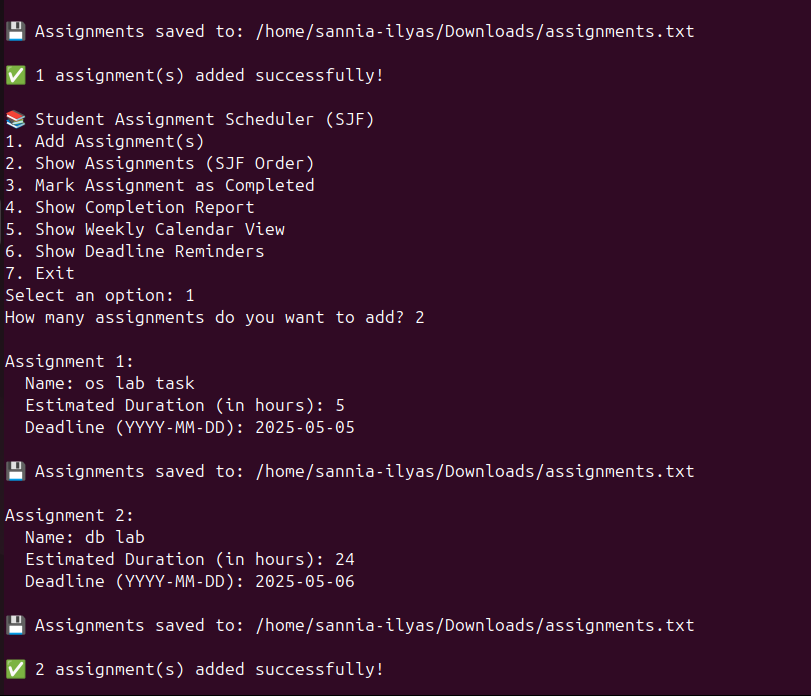
* **Python file :**

****

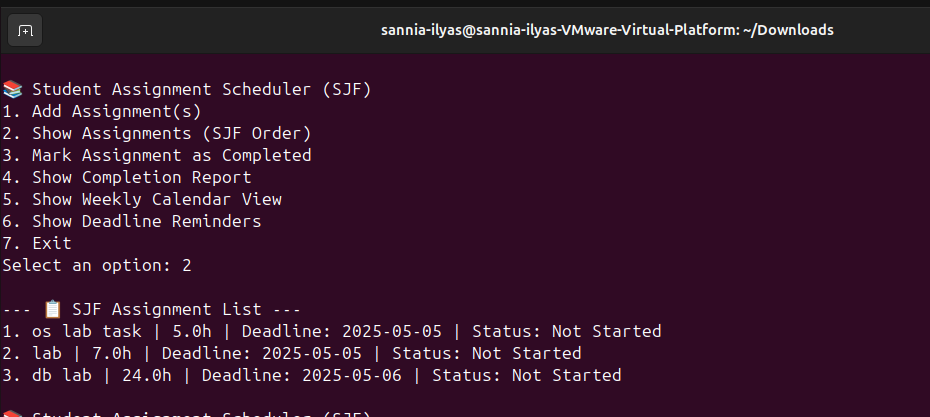
* **Text file where record is saved:**

****

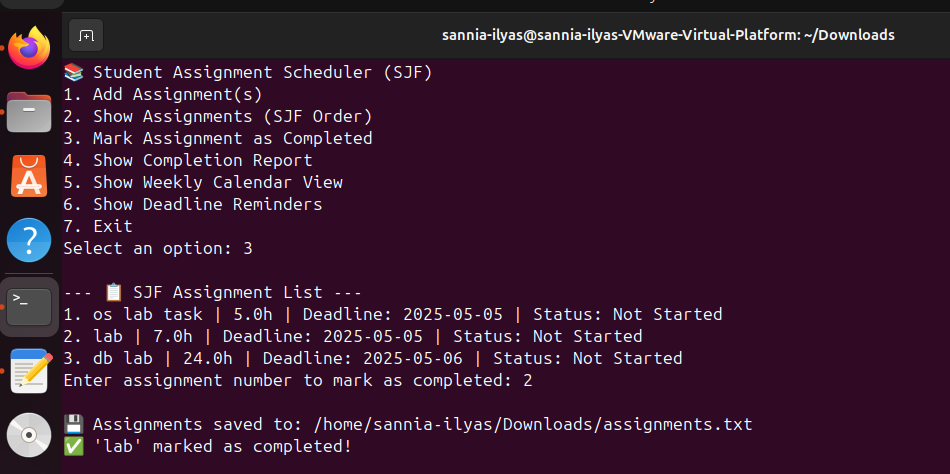
**OUTPUTS:**

**Step 1 **

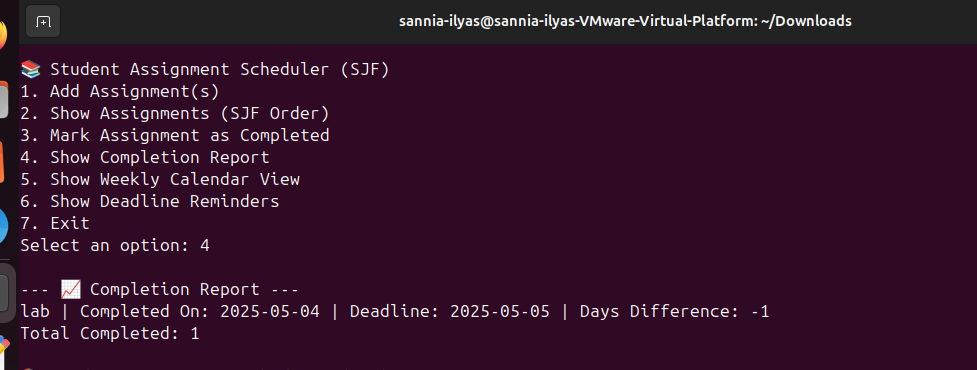
**Step 2**

****

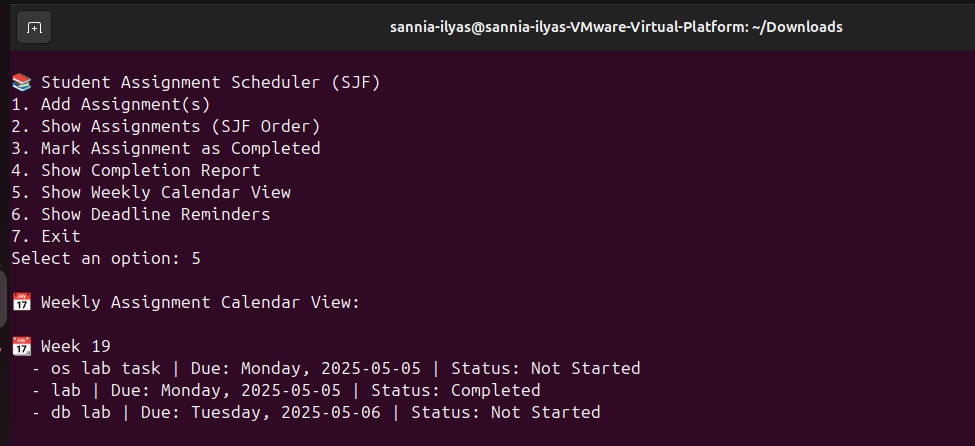
**Step 3**

****

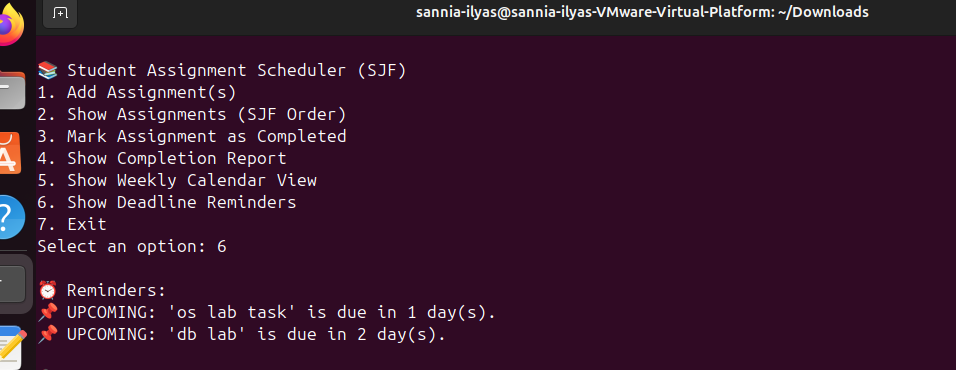
**Step 4**

****

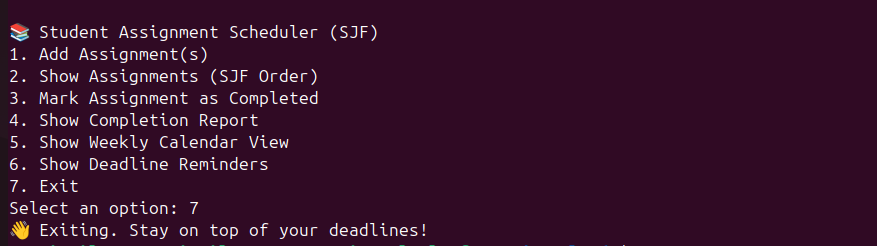
**Step 5**

****

**Step 6**

****

**Step 7**

****