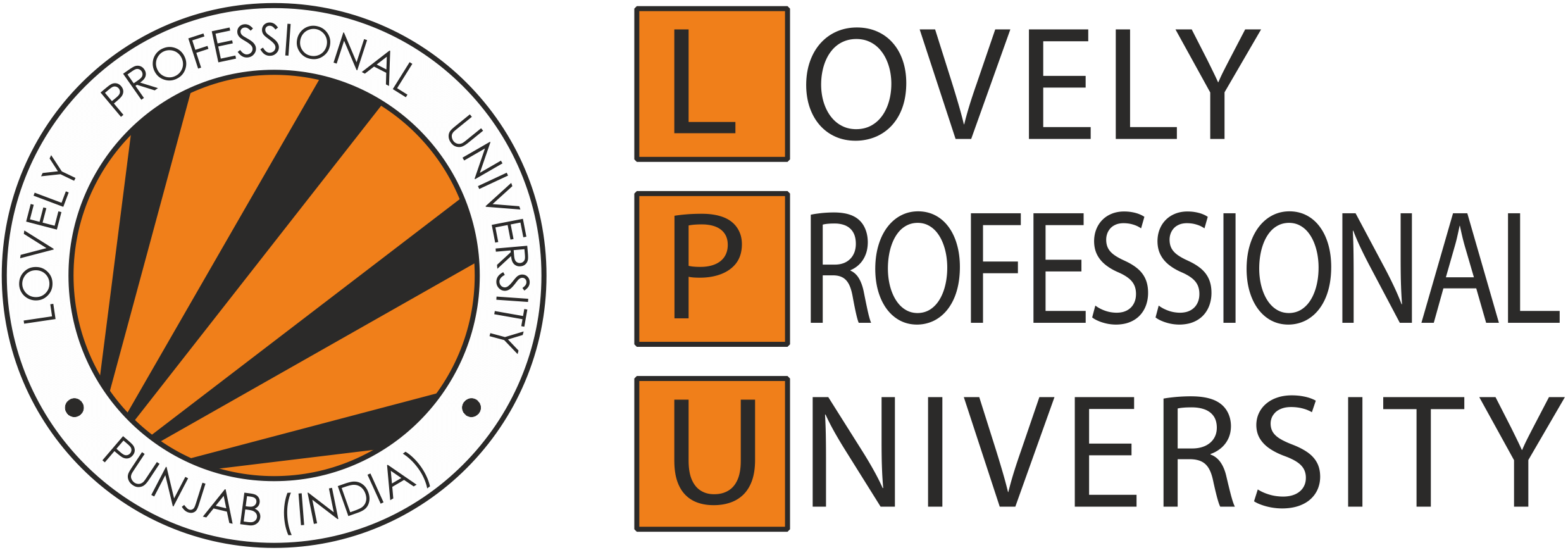
****

**CA-2**

**CSE316: OPERATING SYSTEM**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by: Dhilara Mirsa**

**Registration No.: 12312382**

**Section: K23TG**

**Develop a CPU scheduling algorithm that minimizes energy consumption without compromising performance. The algorithm should be suitable for mobile and embedded systems where power efficiency is critical.**

# **Energy-Aware CPU Scheduling Simulator**

# In this era of smartphones, smartwatches, and IoT devices controlling our lives, energy efficiency is paramount. From a wearable fitness tracker to an embedded system in a drone, each milliwatt of energy counts. That's where this project is useful.The goal of my project was to create a CPU scheduling algorithm that's not only fast — but also smart enough to conserve power. The conventional scheduling algorithms primarily concern themselves with speed and performance. However, within mobile and embedded systems, we have to find a balance between speed and battery life. Therefore, I created a Python desktop application that emulates an energy-conscious CPU scheduler. The application takes user input for different tasks (such as arrival time, burst time, and priority), and schedules them according to a hybrid of Shortest Job First and Priority Scheduling. But here's the catch: it estimates how much energy every task would require, based on its priority. The lower the priority, the higher the power it would require — just like in real life where background processes receive less CPU time. To make it interactive and visually pleasing, I implemented a GUI with Tainter and used Matplotlib to create a Gantt chart that graphically shows how tasks are run over time.

# This project taught me how operating systems handle processes and more importantly how to optimize for power a real-world issue that's growing in importance as devices shrink and become more portable.

**Key Features**

**1. Energy-Aware Scheduling Algorithm**

This project uses a proprietary CPU scheduling algorithm that not only takes performance into account but also computes and optimizes energy usage. The energy usage of each task depends on its burst time and priority, mimicking the way actual embedded systems operate with limited power.

**2. Priority + SJF Hybrid Strategy**

The algorithm is a hybrid of Shortest Job First (SJF) and Priority Scheduling. The tasks with smaller burst time are given preference, and among these, the ones with greater priority (smaller numerical value) are executed first. This balancing helps in decreasing both energy consumption and waiting time.

**3. Graphical User Interface (GUI)**

A neat and easy-to-use GUI developed with Tkinter enables users to enter task information such as process ID, arrival time, burst time, and priority. The interface is straightforward enough for students or non-programmers to use and comprehend OS concepts graphically.

**4. Dynamic Task Management**

It allows multiple task insertions (as of now, 5 demo), runs in simulation mode on their execution, and computes the start time, end time, waiting time, and energy utilized per task at runtime.

**5. Gantt Chart Visualization**

A Gantt chart is shown using Matplotlib, giving the user a vivid timeline of the time each process executes. Users can visualize how the scheduler flow works and even view the execution order and timeframe of each procedure.

**6. Real-Time Energy Calculation**

For each task, the system computes:

Energy = Burst Time × (1 / Priority)

This simulates dynamic voltage and frequency scaling behavior in embedded systems, where tasks with lower priority are allocated fewer resources.

**7. Result Table with Metrics**

Upon execution, a tabulated table shows for every task:

• Process ID

• Arrival Time

• Burst Time

• Priority

• Start & End Time

• Waiting Time

• Energy Consumed

This simplifies analysing scheduling efficiency.

**8. Custom Theming and Clean UI**

The application has a modern, clean dark theme with optimized fonts and colours, so it is both easy on the eyes and pleasant to use — even in presentations.

**9. Error Handling and Validation**

The software has error checks for bad or missing input and informs the user with sensible error messages — making it strong and student-friendly.

10. Educational and Practical Value

Aside from its technical correctness, this tool assists students in seeing CPU scheduling algorithms, performance trade-offs, and how OS-level choices affect energy consumption — making it ideal for labs, assignments, and academic demonstrations.

**Expected Outcomes**

**1. Maximizing Energy Usage per Task**

The scheduler operates in a manner to reduce overall energy expenditure. By prioritizing high-priority and short-burst tasks, it reduces critical tasks' energy usage — which is particularly crucial in mobile and embedded systems where the battery is limited.

**2. Enhanced Scheduling Efficiency**

The hybrid algorithm (Shortest Job First + Priority) minimizes total wait time and maximizes throughput while keeping the system responsive and energy-efficient. The tasks are finished in a systematic and optimal sequence.

**3. Clear Visualization of Task Execution**

Users may see visually how each task was scheduled, for how long, and when and how it began and completed through the Gantt chart. This aids learning, debugging, and analysis.

**4. Realistic Energy Simulation**

Every task is given an energy cost depending on its workload and priority, which makes the output a realistic simulation of how energy-aware systems function. This ties the project to the actual world behavior of CPUs with Dynamic Voltage Scaling (DVS).

**5. Interactive and User-Friendly Experience**

The GUI facilitates straightforward testing of varied task scenarios. Such interactivity supports hands-on learning, critical to grasping Operating System topics such as CPU scheduling.

**6. Summary Data for Performance Measurements**

The output table gives an inclusive summary for each task:

•Start Time

•End Time

•Waiting Time

•Energy Used

The data is helpful in the comparison of the ways various scheduling scenarios affect performance and power consumption.

**7. Pedagogic Utility**

The tool is an educational aid for OS students, presenting intangible concepts such as scheduling and power-aware design in a more concrete and interactive manner through simulation and visual output.

**8. Platform Independence**

Because the project is developed in Python and with common libraries such as Tkinter and Matplotlib, it can be executed on Windows, Linux, or macOS without significant modification — making it portable and flexible.

**Module-Wise Breakdown**

**1. User Input Module**

Purpose:

To accept and verify task information from the user.

Key Features:

• Accepts inputs for:

• Process ID

• Arrival Time

• Burst Time

• Priority (1-5)

• Built using Tkinter Entry widgets

• Verifies all fields are completed and valid integers before proceeding

**2. Scheduler Logic Module**

Purpose:

To implement the energy-aware CPU scheduling algorithm.

Key Features:

• Utilizes a min-heap priority queue (heapq) to schedule tasks

• Applies a combination of Shortest Job First (SJF) and Priority Scheduling

• Computes:

•Start Time

•End Time

•Waiting Time

•Energy Consumption

**3. Energy Calculation Module**

Purpose:  
To calculate energy used by each task based on its priority and burst time.

Energy Formula Used

Energy = Burst Time × (1 / Priority)

**4. Gantt Chart Visualization Module**

Purpose:  
To visualize task execution timelines.

Key Features:

* Uses Matplotlib to plot Gantt chart
* Displays each task with start/end time blocks
* Color-coded task bars with process labels

**5. Result Table Module**

Purpose:  
To display the final scheduling results in a structured format.

Displays:

* Process
* Arrival Time
* Burst Time
* Priority
* Start Time
* End Time
* Waiting Time
* Energy Used

**6. User Interface Module (GUI)**

Purpose:  
To create an interactive and accessible user experience.

Key Features:

* Built using Tkinter
* Dark-themed with enhanced fonts
* Organized layout with input, buttons, results, and charts

**7. Validation Module**

The Validation Module is implemented to make sure that all user input entered in the GUI is valid, meaningful, and within the expected range prior to being processed by the scheduling algorithm

Prior to scheduling any task, the module carries out several checks:

1. Empty Field Check

Makes sure that no input field is empty.

if all(cell.get() for cell in row):

# Continue with parsing the input

2. Type Conversion Validation

Converts the input data from strings to integers. In case a user enters a non-integer (such as letters or symbols), the application will detect the error and display a message.

process = int(row[0].get())

arrival = int(row[1].get())

burst = int(row[2].get())

priority = int(row[3].get())

3. Logical Range Check for Priority

Verifies that the priority value falls within the specified range (1 to 5), with 1 being highest and 5 being lowest.

if priority < 1 or priority > 5:

raise ValueError("Priority should be between 1 and 5")

4. Exception Handling

All invalid input or logic error is captured by a try-except block, and an error message is shown by a popup.

except Exception as e:

messagebox.showerror("Error", f"Invalid input or processing error:

{e}")

5.Threading for Task Execution Simulation

Each task can be executed in an independent thread to mimic real-time, concurrent execution. You may use the threading.Thread class to mimic execution without blocking the GUI.

import threading

def run\_task(task):

# Simulate burst time (e.g., with sleep)

time.sleep(task.burst)

# Update UI with task completion

# Run each task in a thread

for task in tasks:

threading.Thread(target=run\_task, args=(task,)).start()

6. GUI Responsiveness (Non-blocking UI)

Executing the scheduler logic within a thread prevents the GUI from freezing during computation:

def threaded\_schedule():

schedule() # your main scheduler function

threading.Thread(target=threaded\_schedule).start()

**Visualization Module**

The Visualization Module is tasked with graphically displaying how CPU tasks are scheduled and run over time. It employs a Gantt chart to display each task's execution window, indicating when it begins and ends in relation to other tasks.

This module uses the Matplotlib library to create the chart and inserts it into the Tkinter interface through FigureCanvasTkAgg. Every task is represented as a horizontal bar in which:

• The bar's length is equal to the burst time.

• The position on the time axis reflects its start and end time.

• Every bar is marked with the respective process ID for ease of understanding.

**Logging Module**

The Logging Module maintains an internal detailed record of the scheduler's activities and results. It logs important events like task execution sequence, arrival and completion times, waiting time, and energy consumption for every task.

While not always apparent to the user, this module is essential in:

• Debugging: Assists in finding logic errors or inconsistencies in task processing.

• Analysis: Enables closer examination of how scheduling choices are made under various input conditions.

• Transparency: Preserves an historic record of simulations, handy to compare performance for several runs.

**Graphical User Interface (GUI) Module – Explanation**

GUI Module offers a user-friendly front-end for users to enter task data, simulate, and view results. Written in Tkinter, it makes difficult scheduling principles easy through a simple and intuitive layout.

Major duties of the GUI Module are:

• Entry Collection: Users can enter process ID, arrival time, burst time, and priority through Entry fields.

• Control Flow: Buttons such as "Run Scheduler" and "Exit" assist users in simulating the process.

• Results Display: Post-scheduling, results are displayed in an orderly table, indicating task-wise start time, end time, waiting time, and energy consumed.

• Chart Embedding: Embeds a Gantt chart (through Matplotlib) within the interface for easy timeline visualization.

**Input Data Management**

**# Task Input Fields**

o The GUI takes four major parameters for every task: Process ID, Arrival Time, Burst Time, and Priority.

o Process ID is an identification number for every task so that the system can distinguish between them.

o Arrival Time indicates when every task is placed into the queue.

o Burst Time is the time the CPU takes to execute the task.

o Priority (1-5) is the degree of importance of the task, with 1 being the most important.

o The user can input data in a dynamic table that adjusts according to the number of tasks they want to schedule.

o Data is input for each task row-by-row, and the application ensures all rows have data in all fields before moving on.

**# Data Validation**

• Empty Field Check:

o The application prevents fields from being blank when users input task information. If a field is blank, the system informs the user to enter the missing information.

o For instance, the user will be reminded if they omit entering Burst Time or Priority.

if all(cell.get() for cell in row):

# Continue processing

else:

messagebox.showerror("Error", "All fields must be filled out!")

• Type and Format Validation:

no

The system checks that input data types are correct:

 Integer checks that numerical values are input for Arrival Time, Burst Time, and Priority.

 For Priority, the application checks whether it is an integer in the valid range (1-5). If a non-numeric value is input, or if the value is not in range, an error message is shown.

try:

process = int(row[0].get())

arrival = int(row[1].get())

burst = int(row[2].get())

priority = int(row[3].get())

if priority < 1 or priority > 5:

raise ValueError("Priority must be between 1 and 5")

except ValueError as e:

messagebox.showerror("Error", f"Invalid input: {e}")

• Logical Range Validation for Priority:

o The priority values should be between 1 and 5. The system verifies whether the priority falls within this range, and shows an error message if it doesn't.

if priority < 1 or priority > 5:

messagebox.showerror("Error", "Priority must be between 1 and 5")

**# Real-Time Feedback for Users**

• Error Handling:

o If the data entered is invalid or incomplete, the system will detect these errors and give users proper feedback. For example, if a non-integer is entered, or the priority is beyond the valid range, an error message is invoked using messagebox.showerror.

o This makes the user immediately aware of what should be corrected, enhancing the overall user experience.

except Exception as e:

messagebox.showerror("Error", f"Invalid input or processing error: {e}")

**# Data Entry Management**

• Task Table:

o The user can enter data for several tasks using a dynamically created input table.

o Each row represents a task, with columns for Process ID, Arrival Time, Burst Time, and Priority.

o Users can easily add or change tasks by entering data into the corresponding fields, and each task's input is validated before the scheduling algorithm starts.

**# Visual Feedback and Interaction**

• Gantt Chart Visualization:

No After scheduling the tasks, a Gantt chart is presented with task execution timelines.This visual feedback informs users of how tasks were executed over time, as well as how priorities and energy consumption influenced the scheduling.

• Result Table:

o A formated results table shows important indicators for every task, including Start Time, Finish Time, Wait Time, and Energy Consumed. The table is auto-completed after completing the scheduling algorithm, giving the users a plain overview of performance of each individual task.

**# Control Buttons and Functions**

• Run Scheduler Button:

o The user can select the Run Scheduler button after completing all tasks and checking the information to run the scheduling algorithm.

o The button instructs the scheduler to compute task start/end time, waiting time, and energy consumption.

• Exit Button:

o A user can shut down the application with the Exit button, providing a guarantee that the user is able to terminate the program safely.

**Components**

**1.Programming Language**

Python 3.13:

• Implemented for core scheduling logic, energy computation, input checking, and Gantt chart display.

• Supports object-oriented design to handle tasks and scheduling behavior efficiently.

**2. Graphical User Interface (GUI)**

Tkinter

• Manages all GUI elements like input forms, buttons, tables, and layout.

• Gives a visually interactive platform to users to input task data, initiate scheduling, and see results.

**3. Data Visualization**

Matplotlib

• Implemented to plot a Gantt Chart showing task execution timelines.

• Provides visualization of process start/end times and task ordering in a clear and intuitive manner.

**4. Scheduling Algorithm**

• Hybrid model that mixes Priority Scheduling and Shortest Job First (SJF).

• Implemented by utilizing Python's built-in heapq module (min-heap) to keep the ready queue efficient.

**5. Energy Calculation Logic**

• Mimics CPU energy behaviour with the formula:

Energy = Burst Time × (1 / Priority)

• Captures real-world dynamic voltage scaling effects observed in embedded systems and battery-powered devices.

**6. Input Validation & Error Handling**

• Built-in Python error checking, combined with tkinter.messagebox, provides friendly feedback when incomplete or invalid input is provided.

• Prevents crashes and retains robustness at runtime.

**7. Development Environment**

IDEs such as PyCharm / VS Code

• Utilized for constructing, debugging, and testing the software.

• Facilitates code linting, version control integration, and GUI development

**Revision Tracking on GitHub**

**Upload a github link here :**   
-> [dhilaramirza](https://github.com/dhilaramirza)

**Conclusion**

The Energy-Aware CPU Scheduling Simulator is an interactive and educational environment where users can see different tasks scheduled differently in a CPU system with emphasis on power consumption. Since both the Priority Scheduling and Shortest Job First (SJF) approaches are combined within it, the simulator better simulates the real-life CPU environment compared to single-algorithm simulations.

The blend of Tkinter for the GUI, Matplotlib for Gantt chart display, and core logic implementation of Python provides a user-friendly as well as informative experience. Users may provide input for task information, check entries, and display the schedule result together with energy consumption information in real-time.

This tool can function both as an educational tool for understanding operating system concepts and as a starting point for more complex simulations.

**Future Scope**

**1. Support for Multilevel Queue Scheduling**

Add simulation of multiple queues using various scheduling policies for increased realism.

**2. Dynamic Priority Adjustment**

Add algorithms in which task priorities change over time according to wait time, deadlines, or energy consumption trends.

**3. Toggle & Process-Level Simulation**

Add support for simulating real threading behaviour using Python's threading module, including handling deadlock and context switching.

**4. Advanced Visualization Tools**

Add NetworkX integration to display resource dependencies and CPU/thread mapping.Provide timeline zoom, playback control, and visual data exportability for improved analysis.

**5. Battery-Aware Scheduling Model**

Integrate device simulation models running on battery and modify the scheduling to extend battery usage.

**6. Web-Based Interface**

Implement a web-based version with Flask/Django and JavaScript visualization libraries for larger adoption and cloud hosting.

**References**

1. GeeksforGeeks. (n.d.). CPU Scheduling in Operating Systems. Retrieved from <https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/>

2. TutorialsPoint. (n.d.). Python GUI Programming (Tkinter). Retrieved from <https://www.tutorialspoint.com/python/python_gui_programming.htm>

3. GeeksforGeeks. (n.d.). Dynamic Voltage Frequency Scaling (DVFS). Retrieved from <https://www.geeksforgeeks.org/dynamic-voltage-frequency-scaling-dvfs/>

4. Python Software Foundation. (n.d.). heapq — Heap queue algorithm. Retrieved from <https://docs.python.org/3/library/heapq.html>

5. JournalDev. (n.d.). Matplotlib Python Tutorial – Complete Guide for Beginners. Retrieved from <https://www.journaldev.com/20491/matplotlib-python-tutorial>

6. Stack Overflow. (n.d.). How to Create a Gantt Chart in Matplotlib. Retrieved from community discussion: <https://stackoverflow.com>

7. Towards Data Science. (2020). Energy Efficient Scheduling in Embedded Systems. Retrieved from <https://towardsdatascience.com>

8. Real Python. (n.d.). Python GUI Programming With Tkinter. <https://realpython.com/python-gui-tkinter/>

9. Matplotlib Developers. (n.d.). Matplotlib – Visualization with Python.

<https://matplotlib.org/stable/index.html>

**Solution/Code**

A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.