# Notexio — CSE323 (Operating Systems Design) Project Report

Project: Notexio Text Editor (Python + Tkinter)

Course: CSE323 — Operating Systems Design

Version: 1.0.0

Date: 2025-12-20

Author: <Your Name> | Student ID: <Your ID>

## Short introduction (what the project is)

Notexio is a lightweight, customizable text editor built with Python and Tkinter. It was designed as an OS-focused course project to demonstrate practical OS-adjacent concepts such as file I/O, safe persistence, event-driven GUI programming, and basic OS integration (printing, filesystem interactions, configuration storage).

- File operations: new/open/save/save as, recent files, unsaved-change prompts.
- Editing: undo/redo, find/replace, go-to-line, clipboard operations.
- Formatting and view: fonts, colors, zoom, fullscreen, word wrap, line numbers.
- Tools: document statistics, reading time, cleanup utilities.
- Safety: recovery snapshots and optional auto-save.
- OS integration: printing and export as PDF.

## GitHub submission links (required)

GitHub repository: <paste your repo link here>

GitHub Pages link: <paste your GitHub Pages link here>

2–5 minute demo video: <paste your YouTube/Drive link here>

## Project structure (high-level)

- main.py: application entry point, menu bar, shortcut bindings.
- src/file_manager.py: open/save/recent files + unsaved-change workflow.
- src/safety_features.py: recovery files + optional auto-save.
- src/misc_features.py: printing and PDF export, optional drag & drop.
- src/ui_components.py: toolbar, status bar, and line numbers UI.
- Other modules: editor core, edit operations, formatter, view manager, tools, theme and settings managers.

# OS / Systems concepts demonstrated

- File I/O and persistence: UTF-8 reads/writes, path handling, exceptions, saved vs. dirty state.
- Safe shutdown and data-loss prevention: save / don’t save / cancel prompts; recovery-file strategy.
- Threads vs. UI event loops: Tkinter is single-threaded; background timing must hand off widget work to the UI thread.
- Cross-platform OS integration: platform-specific printing and dependency fallbacks.

# Challenges faced (STAR format) — problems and fixes

## Challenge 1 — Auto-save crashes due to unsafe Tkinter access (Thread-safety)

Situation: Auto-save ran on a background thread and intermittently caused freezes or instability.

Task: Implement reliable periodic recovery without blocking the UI or violating Tkinter thread-safety rules.

Action:

- Identified root cause: Tkinter widgets are not thread-safe; calling Text.get() from a worker thread is undefined behavior.
- Scheduled content snapshot on the UI thread via root.after(...).
- Wrote recovery files on a worker thread (disk I/O) and added millisecond timestamps to avoid collisions.

Result:

- Auto-save became stable and predictable.
- UI stayed responsive while still producing recovery snapshots.
- Recovery filename collisions were eliminated.

Theory / notes: Tkinter runs a single-threaded event loop; all widget access must occur on the UI thread. Using after() safely queues work on that loop; disk writes can be offloaded to a worker thread.

## Challenge 2 — Preventing data loss on exit (Unsaved-change workflow)

Situation: Users could lose work if the editor closed without a consistent prompt and save path.

Task: Provide a safe close workflow like real editors (save / don’t save / cancel).

Action:

- Tracked a dedicated is_modified flag (dirty state).
- On new/open/exit, prompted the user and routed the decision to save or cancel.
- Updated UI indicators (title/status bar) so users can see unsaved state.

Result:

- Accidental data loss was prevented.
- Behavior matched user expectations from standard desktop editors.

Theory / notes: Because persistence is explicit (writes happen only on save), the app must gate destructive actions using a reliable dirty-state model.

## Challenge 3 — “Open Recent” menu opened the wrong file (Closure capture bug)

Situation: Dynamic menu generation can accidentally bind all items to the last loop variable value.

Task: Ensure each menu item opens its corresponding file.

Action:

- Used a lambda default argument: command=lambda fp=filepath: open_file(fp).
- Refreshed the menu after open/save to keep the list accurate.

Result:

- Each entry opened the correct file consistently.

Theory / notes: Python closures are late-bound; default arguments evaluate immediately and capture the intended value.

## Challenge 4 — PDF export broke on special characters (Markup escaping)

Situation: ReportLab Paragraph interprets text as markup; characters like &, <, > can break rendering.

Task: Export arbitrary plain text to PDF reliably.

Action:

- Escaped markup-sensitive characters before building Paragraph objects.
- Split the document into lines and added spacing for readability.

Result:

- PDF export worked for normal text and code-like content without errors.

Theory / notes: Escaping prevents the PDF text engine from parsing user content as tags/entities.

## Challenge 5 — Printing is OS-dependent (Windows vs. Linux/macOS)

Situation: Printing uses different OS mechanisms and Python integrations on different platforms.

Task: Support printing where possible and fail gracefully otherwise.

Action:

- On Windows, attempted pywin32-based printing; displayed a clear message when missing.
- On Linux/macOS, printed via system commands (lp/lpr) using a temporary file.
- Added a print preview window to validate content before printing.

Result:

- Cross-platform printing became possible with clear fallbacks.
- The app avoided crashes when dependencies were missing.

Theory / notes: Printing is an OS service; robust apps detect platform capabilities and degrade gracefully with user guidance.

### Challenge 6 — Line numbers drifted during scrolling (UI sync)

Situation: The line-number gutter could desynchronize from the text while scrolling and editing.

Task: Keep line numbers aligned with the text widget’s scroll position.

Action:

- Updated line numbers on modifications and synced scroll via yview() fractions.
- Repacked widgets on toggle to keep layout stable.

Result:

- Line numbers remained aligned during normal editing and scrolling.

Theory / notes: Using yview() as the single source of truth avoids drift caused by multiple scroll inputs (wheel, scrollbar, programmatic).

## Test plan (what I verified)

- Open/save files and verify correct content persisted.
- Modify content and verify the unsaved indicator and prompt-on-exit flow.
- Generate and restore recovery files from the recovery/ directory.
- Export PDF and open it to verify readability.
- Toggle line numbers and scroll to confirm alignment.

## Demo video plan (2–5 minutes, required)

- Show GitHub repo + GitHub Pages page (submission requirement).
- Create/edit a file; show modified indicator.
- Find/Replace and Go-to-Line quickly.
- Recent files menu demonstration.
- Zoom + line numbers + theme toggle.
- Export as PDF and open the result.
- Exit with unsaved changes to show safety prompt.

## Conclusion

Notexio demonstrates OS design concepts through a real GUI application: safe file I/O, graceful OS integration, and reliability-focused features like recovery and safe shutdown. Key lessons were event-driven design, cross-platform behavior, and building safeguards that mirror professional desktop editors.