AFRICAN INSTITUTE FOR MATHEMATICAL SCIENCES

(AIMS RWANDA, KIGALI)

Name: Group 3 | Final Report: 1
Course: PyPro-SCiDaS | Date: October 8, 2025

Working Directory Generator (Project 3)

Group 3 Members:

Josiah Mutie
Belise Kanziga
Djadida Uwituze
Patrick Nizeyimana
Kutlo Gaone Kejang

Abstract

This report presents the design and implementation of the **AIMS-Rwanda Working Directory Generator**, a Python-based automation tool developed to create structured student workspaces for each cohort at AIMS-Rwanda. The program reads two CSV files, a course list and a student list, and automatically constructs a directory hierarchy containing each student's folder and per-course subfolders with placeholder `README.txt` files.

Beyond its technical description, this report explains the motivation behind automating repetitive academic workflows, the methodology used in designing and testing the program, the reasoning behind every key function and library choice, and how such a system can be applied in real-world academic and organizational settings to improve efficiency. The report is written to be understandable even for readers without prior Python experience.

Contents

1. Introduction

The African Institute for Mathematical Sciences (AIMS) Rwanda hosts students from diverse backgrounds and languages, studying multiple advanced courses. Managing digital coursework can become complex when each student manually creates folders and organizes course files inconsistently.

This project aims to solve that issue by developing a simple but robust Python program to automatically generate a clean, uniform workspace for all students in a cohort. The resulting directory structure improves organization, avoids confusion and ensures that every student's work follows a consistent format.

The project is written entirely in **pure Python**, relying only on built-in libraries to comply with the given restriction of not using any external modules such as `pandas` or `numpy`.

2. Objectives

The objectives of the project are to:

- Automatically create a main directory named `AIMS-Rwanda-Workspace`.
- Create subdirectories for each student, using the format `LastName, FirstName`.
- Normalize student names by removing accents, special characters and extra spaces to ensure compatibility with all file systems.
- For each student, create a subdirectory for each course listed in the course CSV file.
- Inside each course folder, generate an empty `README.txt` file.
- Validate the two input CSVs to ensure they exist and are not empty before execution.
- Avoid overwriting existing data; print informative messages instead.
- Provide an interactive menu to rerun the process or exit.

3. Methodology

3.1 Overview of approach

The program follows a modular structure. Each functional aspect, reading CSVs, cleaning names, creating directories and displaying menus, is implemented as a separate function. This design supports clarity, maintainability and easy debugging.

To ensure universal compatibility, we decided to make the system handle special characters, accented names and diverse file encodings such as UTF-8, UTF-8-SIG and Latin-1, Schwarz (2019).

3.2 Libraries used and justification

We restricted ourselves to Python's standard library, carefully selecting modules that best serve each requirement. The information regarding the packages was taken here `https://docs.python.org/3/`:

- **os (Operating System)**: Handles directory and file operations such as checking existence, creating folders and writing files. It ensures portability across Windows, macOS and Linux.
- **csv (Comma Separated Values)**: Reads and parses the student and course CSV files line by line without external dependencies.
- **unicodedata**: Used to normalize names by removing accents (e.g., "García" → "Garcia"), ensuring filesystem-safe folder names.
- **re (Regular Expressions)**: Cleans unwanted characters like punctuation or symbols that could cause filesystem errors.
- **typing**: Provides type hints for better code readability and structure (e.g., `List[str]`).

Each library serves a non-overlapping, essential role, allowing the system to function efficiently while remaining simple and transparent.

3.3 Manual creation of CSV files

Two CSV files were manually prepared for testing:

- `Course List.csv`: Contains one course name per line (e.g., "Introduction to Python Programming", "Statistical Regression", "Physical Problem Solving").
- `Student List.csv`: Contains the last and first names of students separated by a comma (e.g., "Mutie, Josiah").

The assumption is that these CSVs simulate real AIMS cohort data and are simple to modify when a new cohort or updated course list is available.

3.4 Program flow

1. Validate that both CSV files exist and are not empty.
2. Attempt to read them using several encodings in order to handle special characters.
3. Clean and normalize student and course names using Unicode decomposition.
4. Create the main directory (`AIMS-Rwanda-Workspace`) if it does not already exist.
5. For each student, create their folder and subfolders for each course.
6. Within each course folder, create an empty `README.txt`.
7. Display a summary of actions and allow the user to rerun or quit.

4. Implementation

4.1 Encoding-safe reading

The `fallback_read` function ensures that files with accented or special characters (common in multilingual names) are correctly interpreted regardless of the encoding. Instead of hardcoding

a single encoding, this function tries multiple options (`utf-8-sig`, `utf-8`, `cp1252`, `latin-1`) until one succeeds.

```python
def fallback_read(the_path: str) -> List[str]:
    encodings_to_try = ["utf-8-sig", "utf-8", "cp1252", "latin-1"]

    for encod in encodings_to_try:
        try:
            with open(the_path, encoding=encod, newline='') as file:
                lines = file.read().splitlines()
                print(f"✅ Successfully read '{the_path}' using {encod} encoding.")
                return lines
        except FileNotFoundError:
            print(f"⚠️ File not found: {the_path}")
            return []
        except Exception:
            continue  # silently try next encoding

    print(f"❌ Could not read file using any encoding: {the_path}")
    return []
```

Figure 1: Implementation of the encoding-safe file reader (`fallback_read`) function.

4.2 Name normalization

Accented characters or diacritics often appear in international student names. To make folder names portable across operating systems, they must be converted to plain ASCII. This step ensures that no folder name causes errors when transferred between Windows, Linux, or macOS systems. The normalization process removes accents (e.g., "García" → "Garcia") and non-alphanumeric symbols while keeping readability intact.

```python
def change_to_ascii(asc: str) -> str:
    if asc is None:
        return ""
    return unicodedata.normalize("NFKD", asc).encode("ascii", "ignore").decode("ascii")
```

Figure 2: Function for converting names into clean ASCII format.

4.3 Student name cleaning

The cleaning function combines the ASCII normalization step with pattern filtering to eliminate punctuation or symbols that could cause filesystem conflicts. It also ensures proper capitalization of hyphenated and compound names, resulting in consistent, standardized folder names.

```python
def clean_std(p: str) -> str:
    if not p:
        return ""
    p = p.strip()
    p = change_to_ascii(p)
    p = p.replace("'", "").replace("`", "")
    p = re.sub(r'[^A-Za-z0-9 \-]', '', p)
    p = re.sub(r'\s+', ' ', p).strip()

    def hyphen(word):
        return "-".join([x.capitalize() for x in word.split("-")])

    return " ".join([hyphen(x) for x in p.split(" ")])
```

Figure 3: Function for cleaning and formatting student names into safe, standardized forms.

4.4 Workspace creation

This is the core function that integrates all components. It reads the validated CSV files, cleans and normalizes names, and creates the necessary directories and subdirectories. The function also generates a summary of created and skipped folders to provide the user with clear feedback. Each student receives a structured workspace containing course-specific folders and placeholder README files.

```python
def make_workspace(student_csv: str, course_csv: str, root: str = "AIMS-Rwanda-Workspace") -> dict:
    """
    This function creates the complete AIMS-Rwanda workspace structure, including:
    - A main folder
    - Each student's personal folder
    - Course subfolders within each student folder
    - README.txt files inside each course folder

    Input:
        student_csv (str): Path to the student list CSV file (LastName, FirstName format)
        course_csv  (str): Path to the course list CSV file
        root        (str): Name of the main workspace directory (default: "AIMS-Rwanda-Workspace")

    Output:
        dict: A summary dictionary showing counts of folders and files created or skipped.
    """
```

Figure 4: Main function for generating the complete AIMS-Rwanda workspace directory structure.

4.5 Interactive menu system

The final component of the program is an interactive command-line menu that allows the user to run the workspace generator repeatedly without manually restarting the script. This function, named `menu()`, provides a simple text-based interface that displays two options: **(1) Generate new workspace** or **(2) Quit**. When the user chooses option 1, the program requests input file names and an optional root folder name, then automatically executes the workspace creation process using the `make_workspace()` function.

To make the program user-friendly and error-tolerant, the menu was designed to:

- Accept blank inputs and automatically substitute default values (e.g., if no file name is entered, it defaults to “Student List.csv” and “Course List.csv”).
- Handle invalid menu choices gracefully by displaying an informative warning and returning to the main menu rather than terminating the program.
- Catch unexpected runtime errors (such as missing files) and continue the menu loop instead of crashing, ensuring a smooth user experience.

Figure 5 shows the sample menu output as it appears in the console.

```python
def menu():
    """
    This function provides an interactive menu for generating AIMS-Rwanda workspaces.
    Input: User selections made through command-line prompts (e.g., file paths, confirmation choices).
    Output: Runs the workspace creation process based on user inputs and displays a summary in the console.
    """

    while True:
        print("\n========= 🧩 AIMS-Rwanda Workspace Generator =========")
        print("1. Generate new workspace")
        print("2. Quit")

        selection = input("Enter your choice (1 or 2): ").strip()

        if selection == "1":
            # Ask for file names (with defaults)
            student_csv = input("Enter student CSV path [default: Student List.csv]: ").strip() or "Student List.csv"
            course_csv = input("Enter course CSV path [default: Course List.csv]: ").strip() or "Course List.csv"
            root = input("Enter root folder name [default: AIMS-Rwanda-Workspace]: ").strip() or "AIMS-Rwanda-Workspace"

            try:
                # Check if the input CSV files exist
                if not os.path.exists(student_csv):
                    print(f"⚠️ Error: The student file '{student_csv}' was not found.")
                    print("Returning to main menu...")
                    continue  # safely go back to menu

                if not os.path.exists(course_csv):
                    print(f"⚠️ Error: The course file '{course_csv}' was not found.")
                    print("Returning to main menu...")
                    continue  # safely go back to menu

                # Run workspace creation safely
                summary = make_workspace(student_csv, course_csv, root)

                print("\n✅ Workspace generation completed successfully!")
                print("📊 Summary of workspace creation:")
                for key, value in summary.items():
                    print(f"  {key}: {value}")

            except Exception as e:
                print(f"❌ Unexpected error occurred: {e}")
                print("Returning to main menu...")
                continue  # do not crash; return to menu

        elif selection == "2":
            print("👋 Exiting the generator. Goodbye!")
            break

        else:
            print("⚠️ Invalid choice. Please enter 1 or 2.")

menu()
```

Figure 5: Interactive menu displayed in the console, allowing users to generate new workspaces or exit the program.

5. Testing and Results

5.1 Testing setup

To verify correctness, we tested the program using small synthetic datasets. In this case, a student list of 19 names and a course list of 5 courses were used to generate 95 total course folders plus 19 student folders.

5.2 Results

The program successfully:

- Created all expected directories and files.
- Handled accented and hyphenated names gracefully, e.g. 'François, ?ukasz' eventually became just 'Francois, Ukasz'
- Skipped existing directories without overwriting data.
- Displayed clear and user-friendly status messages.

When a file was encoded in an unusual format (such as Windows-1252), the fallback reader automatically adapted, preventing crashes.

6. Discussion

The solution achieved robustness, modularity and usability within the limits of standard Python. Each design decision balanced simplicity with real-world reliability.

Some technical highlights include:

- **Name normalization:** Ensures compatibility across operating systems.
- **Encoding handling:** Avoids read errors caused by special characters.
- **Idempotence:** Existing folders are not recreated, preserving prior data.
- **Menu system:** Allows repeated runs with different CSVs.

Potential improvements include adding:

- Logging mechanisms to record each creation event.
- Error reporting for malformed CSVs.
- Automatic generation of summary files in CSV format.

7. Impact of the Project

The impact of the **AIMS-Rwanda Working Directory Generator** extends well beyond fulfilling project specifications. It highlights how automation, when applied thoughtfully, can streamline repetitive administrative and academic processes, improve digital organization and foster efficient collaboration in diverse environments.

7.1 General impact of working directory generators

Working directory generators are increasingly important in both academic and professional contexts. They provide a consistent structure for data storage and file management, ensuring that projects follow an orderly and predictable hierarchy. This uniformity supports version control, team collaboration and data reproducibility, key pillars in scientific research, education and software development, Smaragdakis and Batory (2000).

By automating directory setup, such systems reduce human error, prevent file duplication, and make it easier to locate relevant resources quickly. They also establish best practices for digital hygiene, a vital skill in data-driven fields where file mismanagement can lead to lost time or corrupted workflows.

7.2 Academic and institutional impact

Within AIMS-Rwanda, the generator fosters a culture of digital organization. Every student starts their work in a uniform environment, which minimizes confusion, prevents inconsistent file naming, and simplifies assignment submissions and grading. Tutors and administrators benefit from reduced manual sorting and improved tracking of student progress.

This consistency promotes reproducibility, one of the cornerstones of scientific integrity, by ensuring that analyses, data and results are properly stored and easily retraceable.

7.3 Skill development and capacity building

The project also has an educational impact on the developers and users alike. Building the system from scratch without external libraries strengthened the team's understanding of core Python principles such as file handling, string manipulation and modular programming.

For students, using the generated workspaces helps cultivate disciplined, systematic approaches to data management and computational problem-solving, skills that extend to all fields of research and industry.

7.4 Wider societal and technological relevance

On a broader scale, tools like this demonstrate how simple automation can enhance efficiency in any data-intensive or collaborative environment, from universities and research centers to corporations and NGOs. They serve as entry points for integrating more advanced digital workflows such as automated backups, synchronization with cloud services or integration into continuous deployment systems Smaragdakis and Batory (2000).

7.5 Sustainability and long-term benefits

By promoting structured digital work habits, such systems contribute to long-term institutional sustainability. They reduce the need for manual file restructuring and ensure that important academic materials are organized and preserved for future reference.

The **Working Directory Generator** thus represents more than a software script, it embodies the growing importance of automation in modern education and professional life, bridging the gap between learning and real-world digital infrastructure.

8. Real-life Applications

The concept behind the AIMS-Rwanda Working Directory Generator extends far beyond academic environments. In research institutions, software engineering teams and data science groups, maintaining a consistent and organized workspace structure is essential for collaboration and version control. Automated directory generation ensures that project files, experiment data and documentation are systematically organized across multiple users or projects. For instance, universities could adapt this system to automatically prepare student lab folders each semester, while companies could integrate similar logic into onboarding scripts that create standardized project environments for new employees. In essence, such a tool minimizes human error, enforces structure and saves time, making it valuable wherever structured file management is required, Smaragdakis and Batory (2000).

9. Conclusion

The AIMS-Rwanda Working Directory Generator successfully fulfills all specified project requirements. It provides a reliable and elegant automation solution for organizing student workspaces. The project demonstrates an effective use of Python's standard library for filesystem automation tasks.

Beyond its immediate application, this project exemplifies clean software design: validating input, handling edge cases and ensuring reproducibility, skills critical to any data science or computational project.

10. Recommendations and Future Work

While this project meets its stated goals, several improvements could make it more versatile and user-friendly in larger or production environments:

- Integrate a simple graphical user interface (GUI) to allow non-technical users to select input files and run the generator visually.
- Package the program as a command-line tool with arguments (e.g., `--student-file`, `--course-file`, `--output-dir`) for easier automation.
- Add logging and configuration options to make it suitable for use in institutional IT systems.
- Extend support for cloud-based storage (e.g., Google Drive, Dropbox) so that folders are created directly in shared workspaces.

These recommendations highlight possible directions for further development rather than immediate project requirements.

References

Schwarz, N. (2019). *A Deep Learning Model for Detecting Sarcasm in Written Product Reviews.* PhD thesis, Master's thesis, Interactive Media.

Smaragdakis, Y. and Batory, D. (2000). Application generators. *Encyclopedia of Electrical and Electronics Engineering.*

Python Software Foundation. *Python 3.12 Documentation.* Available at:`https://docs.python.org/3/`

A. Appendix A: Key code excerpts

See Implementation section for illustrative snippets. A Jupyter notebook with the code is attached with the report submission.