

PASSWORD GENERATOR APP-PYHTON TKINTER GUI

Standalone Password Generator Built with Python's Tkinter Library



June 26, 2025

**INDEX**

Chapter 1: Introduction .................................................. 1

* Background of the topic
* Problem Statement
* Why use python for password generators?
* How password generators work in python
* Application
* Implemented
* Objectives
* Scope of the project

Chapter 2: Literature Review / Existing Systems ...................5

* Proposed enhancements over existing solutions
* Proposed enhancements over existing solutions
* This project mitigates the identified issues by means of

Chapter 3: System Analysis & Design .................................................. 7

* + Requirement Analysis (functional & non-functional)
  + System Architecture / Flowcharts / Diagrams
  + Use Case Diagrams

Chapter 4: Implementation .................................................. 11

* + Tools and Technologies used (languages, frameworks)
  + Explanation of modules
  + Screenshots of working modules

Chapter 5: Results & Discussion .................................................. 14

* + Results obtained
  + Challenges faced
  + Comparisons

Chapter 6: Conclusion & Future Scope .................................................. 17

* Conclusion and future scope
  + Limitations
  + Scope for Improvement

Chapter-07: Additional Sections .................................................. 18

* + References

**PASSWORD GENERATOR APP-PYHTON TKINTER GUI**

**CHAPTER-01**

**TOPIC BACKGROUND**

* **PURPOSE OF THE TOPIC**
* A tool for creating secure, strong, and random passwords is called a password generator.
* These passwords are crucial for preventing brute-force attacks and unwanted access to important information, apps, and online accounts. Creating passwords that are hard to figure out or crack is the aim.
* **PROBLEM STATEMENT**

Making secure passwords that are both difficult to figure out and simple to remember is a challenge for many users. Creating passwords by hand frequently results in weak patterns or password reuse, which raises the possibility of hacking. A program that can swiftly generate strong, random passwords and let users alter the length and complexity of the passwords to suit their needs is required.

* **WHY USE PYTHON FOR PASSWORD GENERATORS?**

Python's ease of use, adaptability, and the availability of tools that make randomization and string manipulation easier make it the perfect language for password generators. Among the main causes are:

* Easy to Use: Python is suitable for developers of all skill levels due to its beginner-friendly syntax.
* Strong Libraries: Modules such as random, string, and secrets allow for the quick creation of cryptographically secure passwords and random characters.
* Customizability: Python enables users to specify the length and complexity of their passwords, including the addition of capital letters, numbers, and symbols.
* **HOW PASSWORD GENERATORS WORK IN PYTHON**
* Character Pool Creation: In Python, a pool of characters is generated by combining capital and lowercase letters, numbers, and special symbols. This is how password generators operate.
* Random Selection: For added security, characters are chosen at random from the pool using functions like random.choice() and secrets.choice().
* Password Output: The password that was generated is either saved for later use or displayed.
* **APPLICATION**
* Using distinct passwords to secure internet accounts.
* Safeguarding private information in both personal and business settings.
* Generating passwords for APIs and it systems automatically.
* **IMPLEMENTED**
* Graphical User Interface (GUI): Tools built using libraries like tkinter to provide a user-friendly interface for password generation.
* **OBJECTIVE**
* Security strength: The main objective is to create strong, unpredictable passwords that are hard to figure out or break. To increase security, these passwords frequently contain a mix of capital and lowercase letters, digits, and special characters.
* Avoiding predictable patterns: Using easily guessed patterns, like common words, consecutive letters, or private information like birthdates, is eliminated by password generators.
* Unique password: They ensure each password is unique, reducing the risk of security breaches across multiple platforms where users might otherwise reuse the same password.
* Customization: Users can choose the password's length and complexity according to their needs or platform-specific limitations.
* GUI Integration: Many password generators include a graphical user interface (GUI) to make them accessible to users who are not comfortable with command-line tools. Libraries like Tkinter are commonly used in Python projects for this purpose.
* Clipboard Integration: Many password generators enable users to copy the created password to their clipboard with only one click in order to improve usability.
* **SCOPE**
* Cybersecurity: In order to create strong, unpredictable passwords that lower the danger of hacking and illegal access, password generators are crucial.
* Personal and Enterprise Use: Individuals can use password generators to secure their online accounts, while organizations can employ them to enforce strong password policies across systems and networks
* Software development: Password generators are frequently included by developers into software systems, websites, and apps that need user authentication.

Password generation can be customized to meet specific requirements, such as length, complexity, or character restrictions.

* Educational Projects: With the use of libraries like Tkinter, password generators are easy-to-learn Python projects that teach students about string manipulation, randomization, loops, and GUI programming.
* Automation: They can automate the process of creating multiple passwords for bulk account creation or system setups.

Integration with clipboard functionality (via libraries like pyperclip) enhances usability by allowing users to copy generated passwords directly.

* Accessibility: GUI-based password generators provide user-friendly interfaces for creating passwords, these applications are accessible to non-programmers.
* Advanced Applications: AI/ML-based password generators analyze and improve password strength using datasets of known vulnerabilities.

**CHAPTER -02**

**Literature review / Existing systems**

* **Existing systems**

There are currently a number of password generation programs and tools available, including offline and web-based ones. In addition to password generation, well-known programs like LastPass, Dashlane, and 1Password also provide storage and autofill capabilities. Websites like Norton Password Generator and Strong Password Generator offer easy access to secure password generation.   
In theory, a lot of these technologies generate characters at random from predefined sets (capital letters, numerals, special characters, etc.). Some additionally provide choices like changing length and complexity, pronouncable passwords, and removing similarcharacters.

Common technologies used include:

* JavaScript for web-based generators
* Mobile apps built with Swift or Java
* Desktop apps using frameworks like Electron or JavaFX
* **PROPOSED ENHANCEMENTS OVER EXISTING SOLUTIONS**

While existing tools are powerful, they often:

* Require internet access or user registration
* Include unnecessary complexity for basic users
* Are closed-source or paid beyond a basic level
* Don't offer a lightweight offline desktop solution for quick use.
* **THIS PROJECT MITIGATES THE IDENTIFIED ISSUES BY MEANS OF:**
* **Providing a minimalistic, standalone desktop application built with Python and Tkinter.**
* **Allowing users full customization options for generating passwords**
* **Designed to be open-source and user-friendly, making it suitable for beginners and offline environments.**
* **Functions entirely offline without storing data, thereby ensuring greater privacy and security for users.**

**CHAPTER-03**

**SYSTEM ANALYSIS & DESIGN**

1. Requirement Analysis

* The application must provide an option for users to define the length of the password they wish to generate.
* Users should be able to select which character types to include, such as:
  + Uppercase letters (A–Z)
  + Lowercase letters (a–z)
  + Numbers (0–9)
  + Special symbols (e.g., @, #, $, %, etc.)
* Upon clicking the "Generate" button, the password should be produced and displayed immediately.
* A feature should be available to copy the generated password directly to the system clipboard.
* The interface must respond accurately to user interactions, including checkbox selections and button presses.
* The system should perform input validation, such as checking for a valid length and ensuring at least one character type is selected.
* A "Reset" or "Clear" option should be provided to reset all fields and selections.
* The app should display clear messages in case of invalid or missing input.
* Optionally, users should be able to toggle password visibility to view or hide the output.
* Simple guidance or tooltips should be integrated to assist users, especially first-time users.

2. Non-Functional Requirements:

* The application should be responsive and user-friendly.
* It should generate the password within one second.
* The system must work offline (no internet dependency).
* The design must be lightweight and run on low-spec machines.
* The application should be secure and must not store any generated password.
* Should have minimal memory and CPU usage.
* The UI should be intuitive, requiring no technical knowledge to operate.
* The codebase should be modular and easily maintainable.
* The application should be platform-independent (should work on Windows, Linux, macOS with Python).
* App startup time should be under 3 seconds.
* No third-party login or user account should be required.

**SYSTEM ARCHITECTURE / FLOWCHART / DIAGRAMS**

1.SYSTEM ARCHITECTURE

**User Interface**

**(Tkinter GUI Window)**

**Input Handler Module**

**(Length, Char Options)**

.

**Password Generator**

**(Random Char Selector)**

**Output / Display Module**

**(Entry box, Copy Btn)**

2.FLOWCHART

**Option to copy password**

**Display password in text box**

**Random password generated**

**Click Generate Button**

**Display GUI Window**

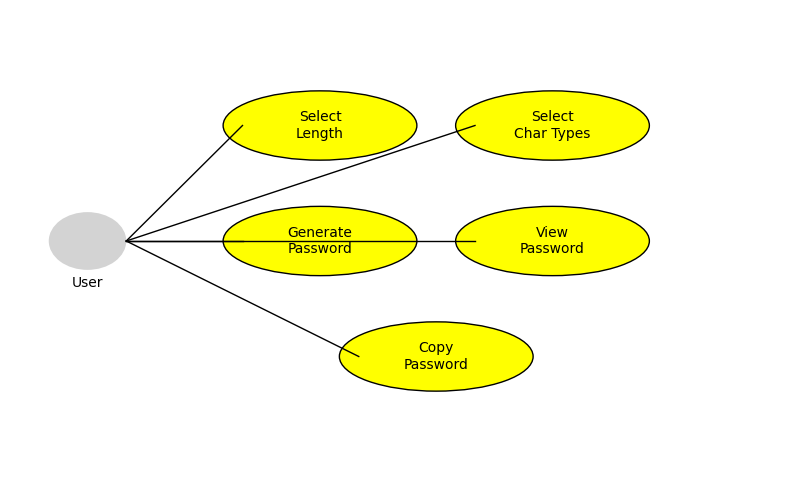
**User selects**

**password options**

**Length**

**Character types**

3.Use Case Diagrams



Functional Use Cases of the Password Generator System

* The Use Case Diagram visually represents the key interactions between the user and the Password Generator App.
* The **user** is the main actor who initiates all actions in the system.

The diagram outlines several key functions the user can perform:

* Select password length based on their security preferences.
* Choose character types such as uppercase letters, lowercase letters, digits, and special characters.
* Generate a password based on the selected criteria.
* View the generated password in the application interface.
* Copy the password for use in other platforms or applications.
* This diagram captures the functional scope of the app, emphasizing usability and security.
* It highlights how the system supports customization and ease of use while maintaining password strength.

**CHAPTER-04**

IMPLEMENTATION

1.PROGRAMMING LANGUAGE

**Python**

* Python is a high-level, versatile, and beginner-friendly programming language.
* It is widely used in application development due to its clean syntax and vast standard library.

**GUI Framework**

**Tkinter**

* Tkinter is Python’s standard GUI (Graphical User Interface) package.
* It is used to design and manage the graphical layout of the application including buttons, checkboxes, labels, and input fields.
* It allows rapid development of user-friendly and interactive desktop applications.

**Supporting Python Modules**

**random**

* Used to generate random selections from the chosen character set for password creation.
* Ensures each password is unique and unpredictable.

**string**

* Provides predefined character sets like ascii\_uppercase, ascii\_lowercase, digits, and punctuation.
* Reduces the need to manually define character pools.

**tkinter.messagebox**

* Provides a simple way to show error or warning dialog boxes to users.
* Helps in validating user inputs and improving user experience**.**

2**.**Explanation of Modules in the Project

1. User Interface Module

* Built using Tkinter, it creates the GUI layout including:
* Title label
  + Entry field for password length
  + Checkboxes for character selection (uppercase, lowercase, digits, symbols)
  + "Generate Password" button
  + Output field to display the result
* Makes the application intuitive and user-friendly

2. Input Handling Module

* Collects input from the GUI elements such as:
  + Password length from the entry field
  + Boolean values for character types from the checkboxes
* Performs basic input validation like ensuring:
  + Password length is numeric and ≥ 4
  + At least one character type is selected

3. Password Generation Logic

* Dynamically builds a character set based on selected options (uppercase, lowercase, digits, symbols).
* Uses random.choice() to randomly select characters and construct the final password of the specified length.
* Ensures the generated password is both random and secure.

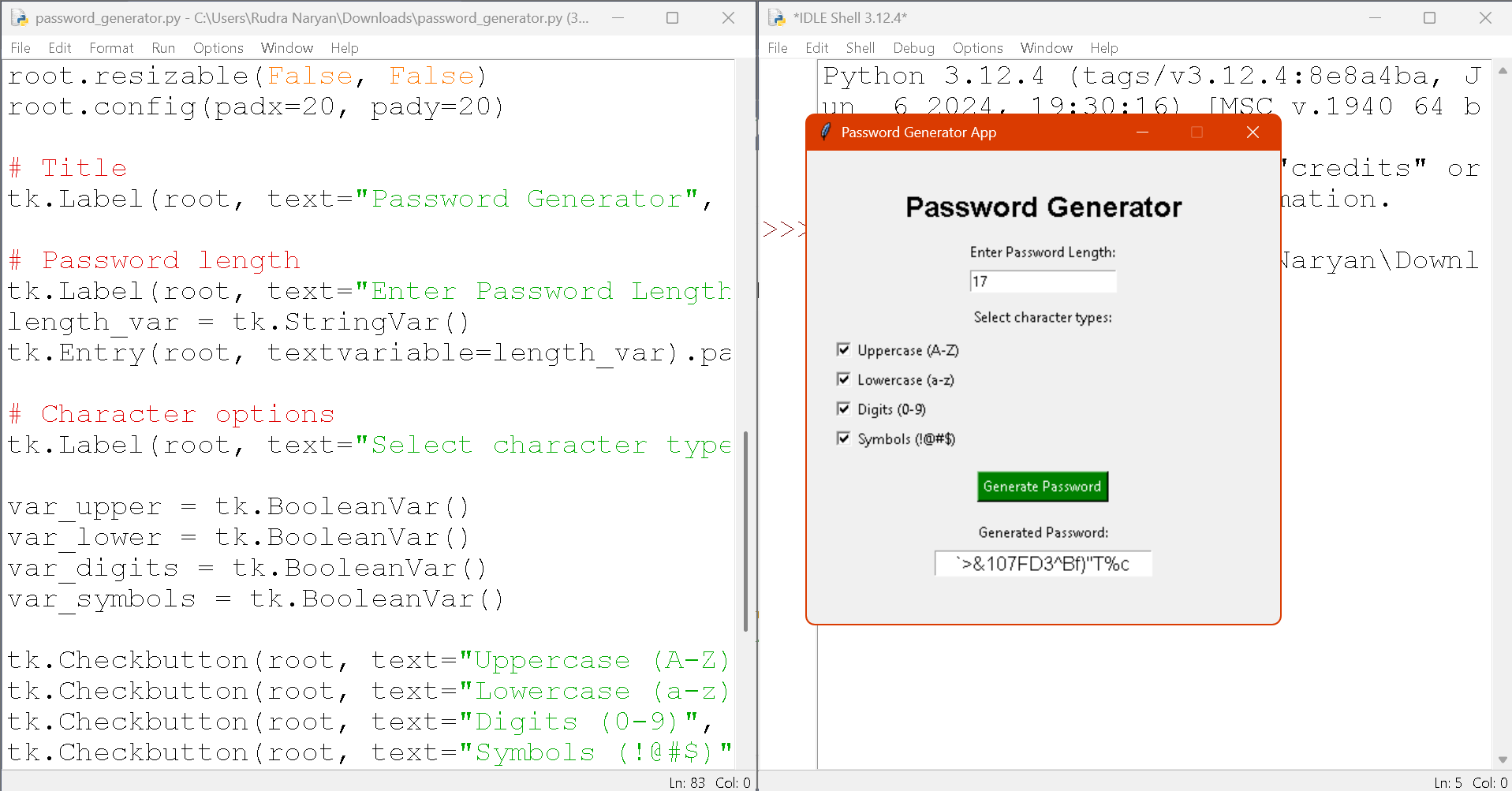
4. Output Display Module

* Clears the previous password from the entry box.
* Inserts the newly generated password into the result field.
* Provides a clear and immediate output to the user after pressing the button.

5. Error Handling & Feedback

* Uses messagebox.showerror() and messagebox.showwarning() for:
  + Invalid input (non-numeric or too short password length).
  + Missing character type selection.
* Ensures users are guided correctly while interacting with the app.

3.WORKING MODULE SCREENSHOT



* **Application Interface Displayed:**
* The screenshot shows the running Password Generator App built using Python Tkinter GUI library.
* **Input Field – Password Length:**
* User has entered the value 17, indicating they want to generate a 17-character password.
* **Checkbox Options for Character Types:**
* The GUI provides four options to customize the password:
  + ✅ Uppercase (A–Z)
  + ✅ Lowercase (a–z)
  + ✅ Digits (0–9)
  + ✅ Symbols (!@#$...)
* All options are selected in this case, ensuring a strong and complex password.
* **Generate Password Button:**
* A green button labeled "Generate Password" is clicked to trigger password creation.
* **Password Output Field:**
* The generated password is displayed in a read-only Entry box.
* Example output shown:

bash

CopyEdit

`>&107FD3^B#]"T%c

* **Code Editor View (Left Side):**
* On the left pane, the Python source code is open in IDLE, showing how the GUI components are created using tkinter.
* **Execution Shell (Right Side):**
* Python version 3.12.4 is visible in the shell where the script is executed.
* **UI Design:**
* The UI is clean and user-friendly, with organized sections for input, options, and output.
* The window is titled "Password Generator App", indicating the project purpose.
* **Security Highlight:**
* The app uses the string and random modules to ensure passwords are generated with good entropy, making them secure.
* **Project Relevance:**
* This screenshot demonstrates successful integration of GUI design, input validation, and dynamic functionality in a Python project.

**CHAPTER -05**

RESULTS & DISCUSSION

* Python's Tkinter module was used to create a Password Generator GUI application that works flawlessly.  
  The app enables users to:
* Enter the length of the password you want.
* Choose between capital, lowercase, number, and symbol characters.
* Create secure passwords at random using predetermined parameters.
* Instantly view and copy the generated password.
* The GUI is responsive, clear, and easy to use.
* Strong, one-of-a-kind passwords that consistently satisfied the chosen requirements were produced.

CHALLENGES FACED

* Putting in place real-time validation (e.g., verifying that no character type was chosen or that input was legitimate).
* Controlling GUI alignment and layout to provide a consistent appearance on various screen sizes.
* Error-checking clipboard operations in older Python/Tkinter versions.
* preventing character duplication or making sure each chosen type has at least one character (not done in the basic version).

COMPARISION

1. Comparison with CLI (Command Line Interface) Password Generators:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aspect | CLI VERSION | |  | | --- | |  |  |  | | --- | | Tkinter GUI Version | |
| User Interface | Text-only; requires typing commands | Graphical; intuitive with checkboxes and buttons |
| Ease of Use | Suitable for programmers | Suitable for all users, including non-technical ones |
| Visual Feedback | No real-time feedback | Immediate visual confirmation and interaction |
| User Experience | Less interactive | Interactive and friendly interface |

2. Comparison with Online Password Generator Tools:

|  |  |  |
| --- | --- | --- |
| Aspect | Online Tools | Tkinter GUI App |
| Internet Dependency | Requires internet connection. | Works offline |
| Security | User data may be exposed to third-party websites. | Local generation ensures data privacy |
| Customization | Often limited or behind paywalls | Fully customizable (length, character types) |
| Ads/Distractions | May include ads/popups | Clean and distraction-free interface |

**CHAPTER -06**

CONCLUSION AND FUTURE SCOPE

* Successfully used Python and Tkinter to create a working password generator
* Created an intuitive graphical user interface (GUI) that enables users to choose the character kinds (uppercase, lowercase, numerals, and symbols) and enter the length of their password.
* Real-time validation has been incorporated to stop user input errors.
* Enabled the ability to immediately generate and display secure random passwords.
* For ease of use, a copy-to-clipboard feature was added.

LIMITATIONS:-

* The password generator does not promise that every password will contain at least one character from every category that was chosen.
* The GUI's simplistic design may make it difficult to adjust to very tiny or large screens.
* Neither managing password history nor saving generated passwords are options.
* The strength of the password (weak, medium, or strong) is not shown graphically.
* lacks user profile or authentication system integration.

SCOPE FOR IMPROVEMENT:-

* Include a password strength meter that changes dynamically according to the length and parameters chosen.
* Use logic to make sure the password has at least one character from each chosen category.
* Improve the GUI with contemporary appearance and responsive design (e.g., using ttk or customtkinter).
* Include a password history feature or the option to safely save passwords.
* Create a web version for cross-platform accessibility with Flask or Django.
* For more sophisticated use, include encryption and integration with cloud-based storage or password managers.

**CHAPTER – 07**

References

1. <https://github.com/banankhanfar/strong-password-generator.git>

2. <https://github.com/nisar-11/The-Password-Manager-GUI-app.git>

3. <https://github.com/TediTae/PasswordManager-GUI.git>

4. <https://github.com/Shruti-175/Password-Generator.git>

5. https://github.com/Ashousha7/Password-Generator.git