Northeastern University
Electrical and Computer Engineering

Password Generator

Author(s): Klaidas Wikar, Jason Zamarripa, Adil Ikiz

Northeastern University

EECE 2140. Computing Fundamentals for Engineers
Fall 2024

Date: Novemeber 22nd, 2024

Abstract

Within this project we sought out the challenge of generating a customizable password securely. Our solution allows users to generate a password by specifying certain features of it such as the length and types of characters to be included. We then send this password to the users inputted email using secure Google servers and the MIME standard. During the development of this program we followed an iterative approach that led to us creating a modular class based system that allowed for clear organization and scalability. In this process we created three classes, PasswordManager, EmailManager, and PasswordEmailManager. These classes respectively handle password generation, email compilation, and class integration. Within this program we also use key python libraries such as random, string, but we also less well known libraries or packages such as smtplib and MIME. The latter allowed us to integrate the email functionality into our code. Combining all these aspects together allow us to create a program that was secure and easily iterated. Our project successfully meets its primary objectives and our own goals we also achieved , however there were certain limitations. These limitations include the absence of a password storage system, and advanced authentication/encryption methods. enhancements could've also been made such as adding a GUI or also adding SMS functionality. These features could've been added but given time constraints we were unable to. Overall this project addressed the need for secure customizable passwords while working with certain limitations.

Contents

Chapter 1

Introduction

1.1 Background

Creating a different password for every website or subscription can be difficult. Having a place where all your passwords are stored can also be difficult. This project will allow the user to get a different password every time and have it saved to an email only they can open.

1.2 Problem Statement

Find a way to create a random password with user defined specifications and get it to the user without showing others the output.

1.3 Objectives

1.Allow the user to get a random password.
2.The user should be able to select what length for the password.
3.The user should be able to select what characters they want in the password.
4.The user should be able to get an output sent to their email.
5. Use the class system to organize the code and allow better security.

1.4 Scope

1.4.1 What is covered:

Password Generation Features: The password generator uses the random library offered through python creating random characters types ever time.
Customization Options: The password generator is able to allow the user to request wether they want letters, numbers, or symbols in their password.
Email Functionality: We are able to have the generated password sent to any

g mail that is on the google g mail servers.
Security and Usability: The password is random every time which allows the password generator to be used unlimited times.

1.4.2 What is not covered:

Advanced Password Management: so our password generator does not store the password in a private file or contain any retrieval method if password is forgotten.
Complex Email Features: Our code is not able to support any complex formatting beyond basic text in the email.
Real-Time User Authentication: Our password generator has no way of authenticating the user as the owner of the email address.

Chapter 2

Technical Approaches and Code UML

2.1 Development Environment

The file"PasswordGen_v03.py", which is the only executable Python file (version 3.12.4), was compiled using the Python IDE "Spyder" (version 5.5.1) through the GUI navigation tool, "Anaconda Navigator" (version 2.6.3); with iterative editions compiled directly through GitHub, with the final edition of the code superficially re-uploaded through GitBash to the master branch of the same repository.
The libraries made use of in the code include the following

- "string" for basic string functionality,
- "random" for random number generation,
- "os" for setting a working directory to the folder the script is in
- "smtplib" along with the modules "MIMEMultipart" and "MIMEText" for email functionality.

2.2 Data Collection and Preparation

The data collected and analyzed in preparation for this project was primarily qualitative in nature. In our search for an appropriate means of engaging the user with the generation of their password, a brief look into how password generation is handled in various platforms suggested that there was primarily two types of information taken into consideration: the complexity and the length of the text.
In this context, complexity in passwords are managed through the use of special characters, as well as digits to go along with the regular set of 26 characters that make up the English alphabet.

2.3 Implementation Details

The development of the code followed an iterative approach, with additional functionalities being added as the code grew. While in summary, it could be said that the code came in three distinct versions: _v01, _v02, and _v03, it is most appropriate to think of the three iterations as one big work-in-progress; where at each stage the following main criterion was achieved and _v03 is the final product:

- Version 01: General functionality of password generation with options to adjust length and complexity of the password.
- Version 02: Email functionality reformating of code under classes
- Version 03: Re-upload through GitBash (master branch)

On a, deeper, step-by-step basis, the compiling of the code consisted of the following stages:

1. (Version 01) Method generation for determining user preferences:
    (a) The inclusion of letters, special characters, and digits are handled by respective boolean values which are determined through user inputs.
2. Method for Password Generation
    (a) Password length is determined through integer entry by the user
    (b) From what preferences were made in step 1, a string of usable characters are created by joining strings that include appropriate character banks. A randomized combination of characters from this combined string at the given length is then generated with the use of random and string libraries. This string is then returned.
3. (Version 02) Method Generation for Email compiling & Sending
    (a) Setting up MIME with sender/receiver info & email subject/body
    (b) Establishing connection to Google Gmail Services through SMTPLib with provided credentials
    (c) Sending of Email & Exception handling
4. Rework on user preference determination method
    (a) Added error handling (user must state YES [1]or NO [2] to each option)
5. Adding of HTML functionality to the email methods (handled by Email-Manager class [see below])

(a) This allows for further stylizing of the output email, making it more appealing a usable on the customer's end; The generated password is now stitched into a template email, which is sent as the final output

6. Rework on entire executable (classification of methods)

   (a) class PasswordManager handles methods relating to user preferneces. Attributes include: pw, letters_bool, special_bool, numbers_bool. Methods include: user_preferences, and generte_password.

   (b) class EmailManager handles methods relating to the compiling of HTML file with generated password, compiling and sending of the email using SMTP. Attributes include sender_email, sender_password, recipient_email, subject, and server. The send_email method handles the actual communications

   (c) class PasswordEmailManager handles the communication between the PasswordManager class methods along with the EmailManager mehtods, allowing for intuitive modularity. Attributes handled include instances of the previous two classes as password_manager and email_manager (initialized with appropriate parameters). The execute method calls necessary methods through these instances for the generation of the password and the sending of the email

Commenting on the code was undertaken throughout the development, as well as a part of routine overview and adjustments.

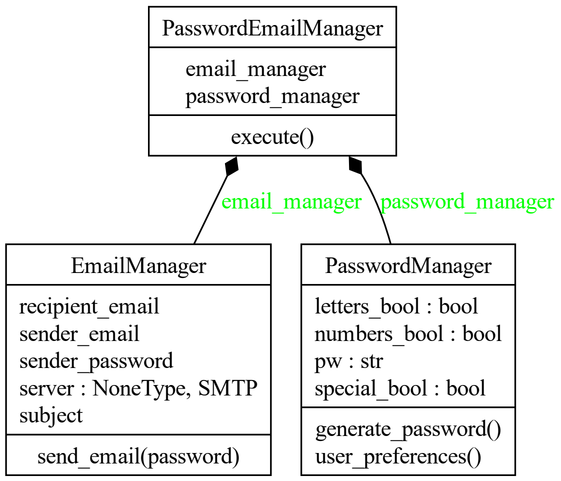
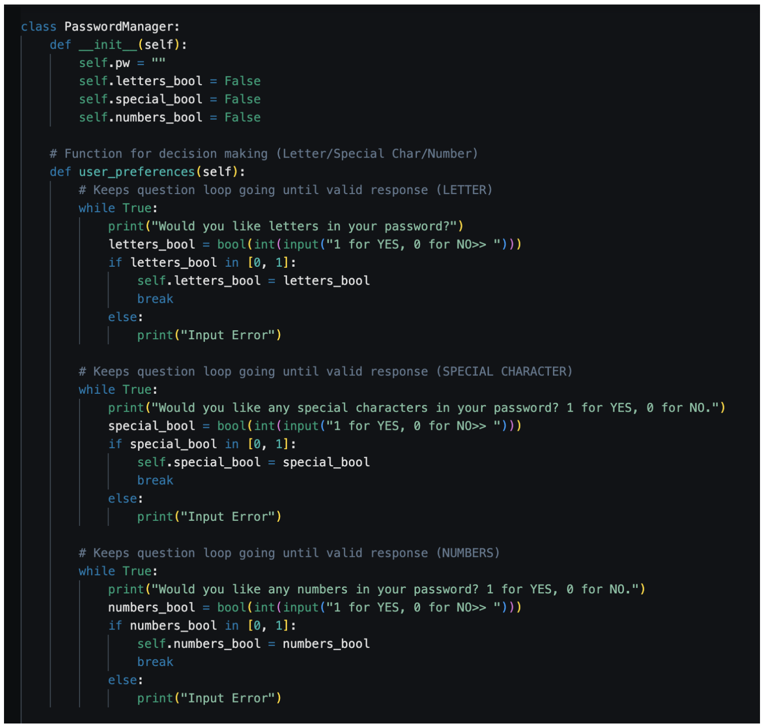


Figure 2.1: UML Diagram of Classes within PasswordGen_v03.py

Chapter 3

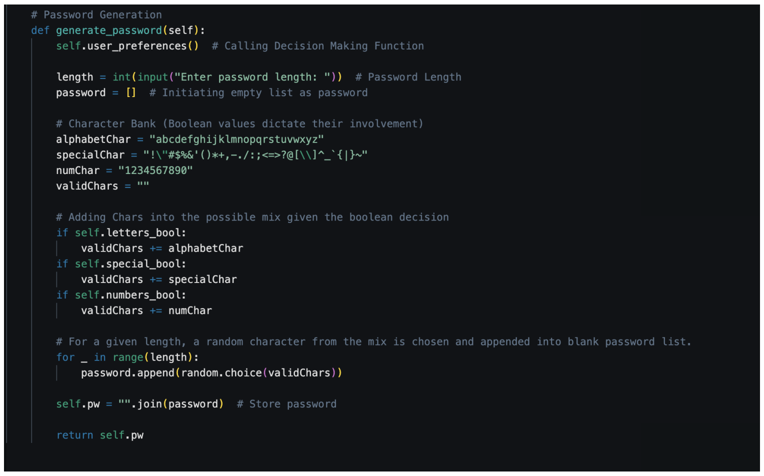
Project Demonstration

3.1 Screenshots and Code Snippets



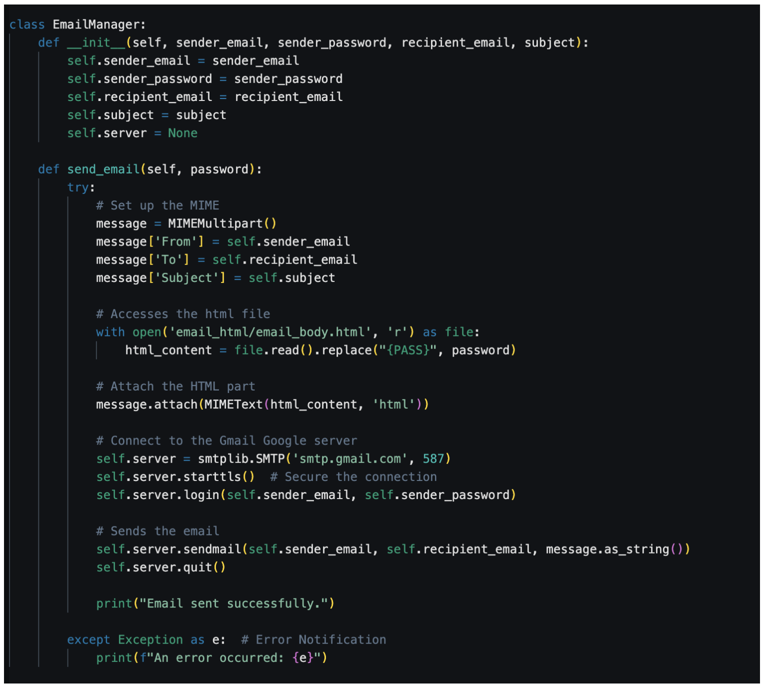
```
class PasswordManager:
    def __init__(self):
        self.pw = ""
        self.letters_bool = False
        self.special_bool = False
        self.numbers_bool = False

    # Function for decision making (Letter/Special Char/Number)
    def user_preferences(self):
        # Keeps question loop going until valid response (LETTER)
        while True:
            print("Would you like letters in your password?")
            letters_bool = bool(int(input("1 for YES, 0 for NO>> ")))
            if letters_bool in [0, 1]:
                self.letters_bool = letters_bool
                break
            else:
                print("Input Error")

        # Keeps question loop going until valid response (SPECIAL CHARACTER)
        while True:
            print("Would you like any special characters in your password? 1 for YES, 0 for NO.")
            special_bool = bool(int(input("1 for YES, 0 for NO>> ")))
            if special_bool in [0, 1]:
                self.special_bool = special_bool
                break
            else:
                print("Input Error")

        # Keeps question loop going until valid response (NUMBERS)
        while True:
            print("Would you like any numbers in your password? 1 for YES, 0 for NO.")
            numbers_bool = bool(int(input("1 for YES, 0 for NO>> ")))
            if numbers_bool in [0, 1]:
                self.numbers_bool = numbers_bool
                break
            else:
                print("Input Error")
```

Figure 3.1: PasswordManager Class



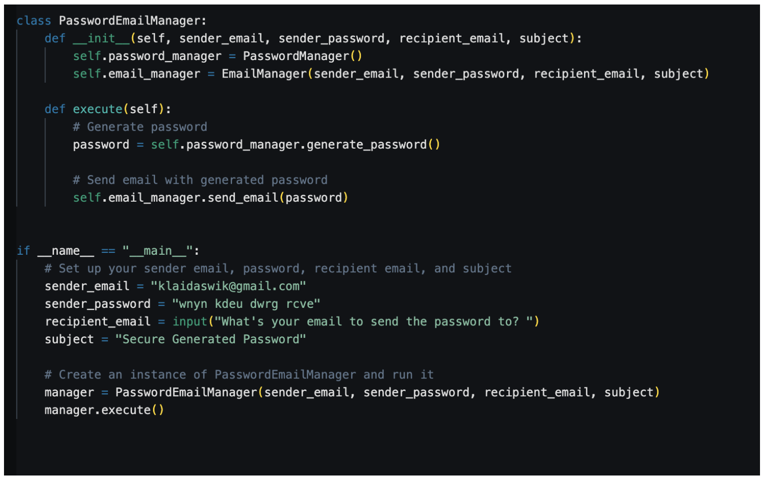
```python
    # Password Generation
    def generate_password(self):
        self.user_preferences()  # Calling Decision Making Function

        length = int(input("Enter password length: "))  # Password Length
        password = []  # Initiating empty list as password

        # Character Bank (Boolean values dictate their involvement)
        alphabetChar = "abcdefghijklmnopqrstuvwxyz"
        specialChar = "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"
        numChar = "1234567890"
        validChars = ""

        # Adding Chars into the possible mix given the boolean decision
        if self.letters_bool:
            validChars += alphabetChar
        if self.special_bool:
            validChars += specialChar
        if self.numbers_bool:
            validChars += numChar

        # For a given length, a random character from the mix is chosen and appended into blank password list.
        for _ in range(length):
            password.append(random.choice(validChars))

        self.pw = "".join(password)  # Store password

        return self.pw
```

Figure 3.2: PasswordManager Class Extended



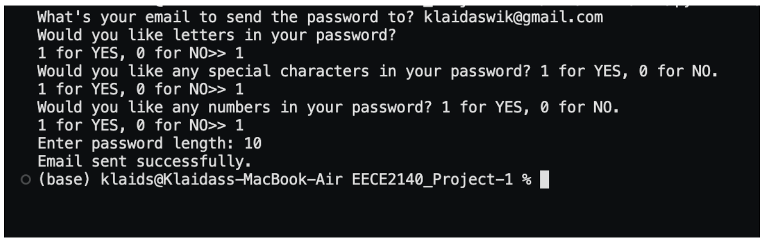
```python
class EmailManager:
    def __init__(self, sender_email, sender_password, recipient_email, subject):
        self.sender_email = sender_email
        self.sender_password = sender_password
        self.recipient_email = recipient_email
        self.subject = subject
        self.server = None

    def send_email(self, password):
        try:
            # Set up the MIME
            message = MIMEMultipart()
            message['From'] = self.sender_email
            message['To'] = self.recipient_email
            message['Subject'] = self.subject

            # Accesses the html file
            with open('email_html/email_body.html', 'r') as file:
                html_content = file.read().replace("{PASS}", password)

            # Attach the HTML part
            message.attach(MIMEText(html_content, 'html'))

            # Connect to the Gmail Google server
            self.server = smtplib.SMTP('smtp.gmail.com', 587)
            self.server.starttls()  # Secure the connection
            self.server.login(self.sender_email, self.sender_password)

            # Sends the email
            self.server.sendmail(self.sender_email, self.recipient_email, message.as_string())
            self.server.quit()

            print("Email sent successfully.")

        except Exception as e:  # Error Notification
            print(f"An error occurred: {e}")
```

Figure 3.3: EmailManager Class



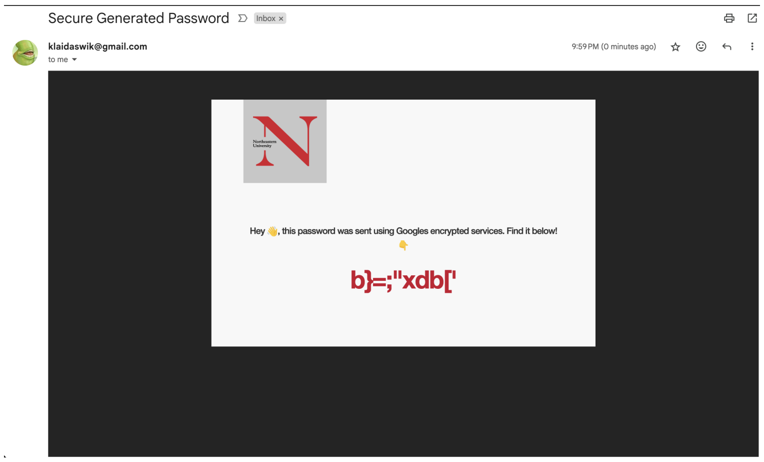
```python
class PasswordEmailManager:
    def __init__(self, sender_email, sender_password, recipient_email, subject):
        self.password_manager = PasswordManager()
        self.email_manager = EmailManager(sender_email, sender_password, recipient_email, subject)

    def execute(self):
        # Generate password
        password = self.password_manager.generate_password()

        # Send email with generated password
        self.email_manager.send_email(password)


if __name__ == "__main__":
    # Set up your sender email, password, recipient email, and subject
    sender_email = "klaidaswik@gmail.com"
    sender_password = "wnyn kdeu dwrg rcve"
    recipient_email = input("What's your email to send the password to? ")
    subject = "Secure Generated Password"

    # Create an instance of PasswordEmailManager and run it
    manager = PasswordEmailManager(sender_email, sender_password, recipient_email, subject)
    manager.execute()
```

Figure 3.4: PasswordEmailManager Class



```
  What's your email to send the password to? klaidaswik@gmail.com
  Would you like letters in your password?
  1 for YES, 0 for NO>> 1
  Would you like any special characters in your password? 1 for YES, 0 for NO.
  1 for YES, 0 for NO>> 1
  Would you like any numbers in your password? 1 for YES, 0 for NO.
  1 for YES, 0 for NO>> 1
  Enter password length: 10
  Email sent successfully.
○ (base) klaids@Klaidass-MacBook-Air EECE2140_Project-1 %
```

Figure 3.5: User Input Screen



Secure Generated Password Inbox ×

klaidaswik@gmail.com
to me

9:59 PM (0 minutes ago)

Hey 👋, this password was sent using Googles encrypted services. Find it below!
👇

b}=;"xdb['

Figure 3.6: Email output

Chapter 4

Discussion and Future Work

4.1 Discussion

Creating a new password for different applications is challenging and many users have a single password for all. If someone is able to retrieve that password, a person's personal data will be breached. Our project encourages users to create different passwords instead of having one password for all. We are able to successfully generate a custom password created for the user's specific needs. The password will never be the same so if one password is breached, every other application is still safe and secure.

4.2 Future Work and Improvements

4.2.1 GUI

A simple to navigate GUI can be adopted to further facilitate password generation. At its current state, the tool provides full functionality at the click of only a handful buttons, however, a further refinement with the following input elements can be proposed:

- Radio type input for selecting which characters to include in the password: letters, characters, and/or numbers
- Number type input for adjusting the length of the password
- Email type input for providing destination email address

- Button type input to act as an execution button, for the generation and sending of the password to the provided email address.

4.2.2 Complex Features

SMS messages: Basic SMS functionality as a further accessibility measure. This functionality would negate the need for internet connectivity, and allow for the user to store the passwords on their mobile devices.

Password Storage: A means to store generated passwords either in an online or an offline fashion would enable the user to essentially create a digital ledger their many passwords for easier subsequent referencing.

Browser Extension: An integration of the functions of this software with the user's browser of choice would allow for seamless password generation for the all of the user's online needs. The choices the user is given in including special characters and digits along with letters as well as changing the length of the password will ensure that the product will have extended usability in most scenarios.

AI Recommendations: Have an AI suggest complexity and length for different applications. For example, a password for a website compared to a password for a bank account. The need for complexity is an important topic when dealing with personal data like a bank account.

Chapter 5

Conclusion

In this project we developed a password generator that creates random, customizable passwords and sends them securely to users via email. By incorporating user defined settings for different aspects like password length and character type. The email functionality, which uses secure Google servers ensures that the generated passwords are kept confidential and delivered in an encrypted way. We were able to implement the password generation systems using pythons random library. We were then able to send the email via pythons SMTP library and the MIME package. The use of a class-based structure in our code creates an organized and scalable program, while also keep the password data secure. If given the proper time and resources we could've expanded our scope to include a storage system of all a users passwords and also include an advanced authentication/encryption method for that storage system. It would also be a nice quality of live improvement for the user if we were able to create a GUI for our program. Overall, this project provides secure and easy to use solution for created and receiving passwords.

Bibliography

[1] WeRoot. *How Do I Create a Strong and Unique Password?*. https://www.webroot.com/us/en/resources/tips-articles/how-do-i-create-a-strong-password

[2] Ciro Santilli. *How to send an email with Python?*. https://stackoverflow.com/questions/6270782/how-to-send-an-email-with-python