OTP Verification System Using Python

M.NAGA KEERTHI, RAIDU SAI SIRISHA
Assistant Professor, Department Of MCA, MCA Final Semester,
Master of Computer Applications,
Sanketika Vidya Parishad Engineering College, Vishakhapatnam, Andhra Pradesh, India

Abstract:

This OTP verification system provides a straightforward method for authenticating users through email-based verification. It begins by prompting the user to enter their email address, then generates a random six-digit one-time password (OTP). The OTP is "sent" to the user's email, simulated by a print statement in this implementation. The user is then asked to enter the OTP they received. To enhance security, the system allows a maximum of three attempts to input the correct OTP. If the user enters the correct OTP within these attempts, access is granted, confirming successful verification. If the user fails after three tries, access is denied to prevent unauthorized entry. The system also includes input validation to handle non-numeric or invalid OTP entries gracefully. By combining randomness, limited attempts, and email-based delivery, the system ensures that only legitimate users can complete the verification process. While this example simplifies email sending, the structure can be extended to real-world email services. Overall, this implementation serves as a basic, yet effective, framework for OTP-based user authentication. It highlights essential security practices, including verification, limited retries, and user feedback during the process.

Index Terms: OTP, Authentication, Python, Security, Email Verification, Random Number Generation, User Validation.

1.Introduction:

In today's digital landscape, safeguarding sensitive information and preventing unauthorized access have become critical priorities. One-Time Password (OTP) systems are a widely adopted method to enhance user authentication during logins or secure transactions.

Unlike traditional authentication methods that rely on static passwords often vulnerable to phishing, brute-force attacks, and repeated use across multiple platforms an OTP system generates a unique, temporary code for each verification attempt. This greatly reduces the risk of credential compromise.

The project presented here implements a straightforward Python-based OTP verification mechanism. It prompts the user to provide an email address, generates a random 6-digit OTP, and validates the entered code within a predefined number of attempts to ensure secure and reliable access control.

1.1. Existing system

Right now, OTP (One-Time Password) verification is often done manually or using very basic scripts. Some systems just send a static code or rely on pre-set passwords, which can be guessed or reused. Others may send OTPs but without limits on the number of attempts, making them vulnerable to brute force attacks. These approaches might work, but they're not very secure or user-friendly.

In short, the existing ways to verify OTPs have been a bit basic and might not always protect against real threats. With a proper automated system like the one in our code, we can generate unique OTPs, send them to the correct email, and limit the number of attempts making the process much smarter and safer, like having a security guard who recognizes you instantly instead of asking the whole neighborhood.

1.1.1.Challenges

OTP Generation and Randomness

- The system relies on Python's random.randint() for OTP creation, which, while sufficient for basic use, may not be cryptographically secure for high-stakes applications. Weak randomness can increase the risk of OTP prediction by attackers.
- Ensuring the OTP has the correct length (6 digits) consistently, without leading zeros being dropped, requires careful handling in formatting and generation.

Email Delivery and Reliability

- Sending OTPs via the send_otp_email() function currently uses a print statement for demonstration. In production, actual email delivery introduces challenges like email server configuration, spam filtering, and delivery delays that can frustrate users.
- Failure in email delivery without proper retry or notification mechanisms can leave users unable to complete verification.

User Input Validation and Error Handling

- The system depends on correct formatting of user email and OTP input. Mistyped emails or non-numeric OTP entries can break the flow if not validated rigorously.
- Handling invalid input gracefully, without allowing repeated abuse (e.g., automated guessing), is essential to maintain both security and usability.

Security Against Brute Force Attacks

- The current 3-attempt limit helps reduce guessing attacks but may still be susceptible if OTPs are short-lived or guessable.
- More sophisticated protections, such as account lockouts, time delays between attempts, or IP-based throttling, require additional implementation.

User Experience and Interface Flow

- The text-based interface may be confusing for non-technical users, especially if error messages are vague.
- Providing clearer prompts, dynamic guidance, and integration with graphical user interfaces could improve accessibility and reduce user mistakes.

1.2 Proposed system:

We want the computer to handle OTP verification in a secure and automated way. Instead of relying on static passwords or basic manual checks, this system will generate a unique 6-digit OTP, send it to the user's email, and verify it within a limited number of attempts. By doing this, it can make the login process both safer and faster. It can be enhanced to use stronger random number generators, integrate with real email services, and add security features like time-based expiration and retry limits, making sure access control is always reliable and up-to-date

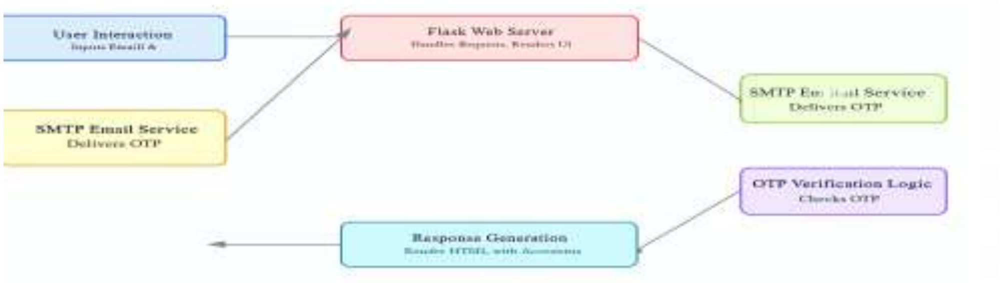


Fig: 1 Proposed Diagram

1.1.1 Advantages:

1. Enhanced Security

- Generates a unique, random 6-digit OTP for each verification attempt.
- Reduces the risk of unauthorized access compared to static passwords.
- Limited number of attempts helps protect against brute-force attacks.

2. Instant Verification

- Users receive OTPs immediately via the email delivery process.
- Eliminates the need for manual identity checks or long approval times.

3. User-Friendly Interaction

- Simple, step-by-step prompts guide the user through entering their email and OTP.
- Minimal technical knowledge required — even non-technical users can complete verification easily.

4. Automates Traditional Security Checks

- Replaces manual password confirmation or identity questioning.
- Reduces human error and speeds up the authentication process.

5. Adaptability and Scalability

- Code can be extended to include features like OTP expiration timers, IP-based restrictions, and SMS delivery.
- Can integrate with more advanced email services or APIs for better reliability.

6. Lightweight and Easy Deployment

- Written in Python, which is platform-independent and easy to maintain.
- Can run on any local machine, server, or cloud environment with minimal setup.
- Scalable to handle increasing numbers of verification requests.

2.1 Architecture:

The architecture of the OTP verification system follows a straightforward but secure authentication pipeline, beginning with the user entering their email address through the program's input interface. This email is used as the destination for sending a unique, randomly generated 6-digit OTP created by the generate_otp () function. The system then triggers the send_otp_email () function (simulated in the current code with a console print statement, but extendable to real email services) to deliver the OTP to the user.

Once the OTP is sent, the system enters a verification loop where the user is prompted to input the received code. This input is processed by the prompt_user_for_otp () function, which validates that it is a properly formatted numeric value. The verify_otp() function then compares the entered OTP with the originally generated one. The process allows up to three attempts to enter the correct code, reducing the risk of brute-force guessing.

If the OTP matches, access is granted, and the verification process ends successfully. If all attempts fail, access is denied, ensuring that the system enforces strict authentication rules. This architecture provides a secure, scalable, and user-friendly method for confirming user identity before granting access to protected resources or services.

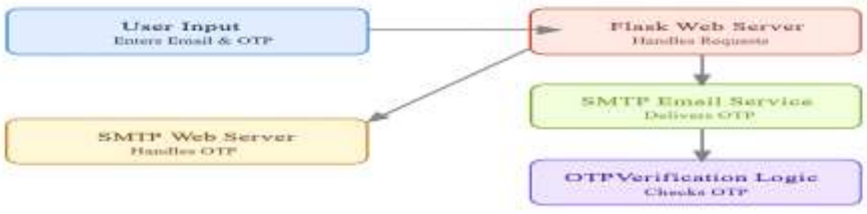


Fig:2 Architecture

2.2 Algorithm:

For implementing OTP verification, several approaches can be used to securely authenticate users. The simplest and most direct method, as in this system, is Random OTP Generation using Python's random.randint() function to produce a 6-digit code between 100000 and 999999. This ensures that the OTP is always numeric, easy for users to enter, and has a sufficient range to reduce the probability of guessing. This approach is computationally efficient and works well for basic authentication needs. However, for high-security applications, cryptographically secure OTP generation methods (such as Python's secrets module or Time-based One-Time Passwords — TOTP) can be used to prevent prediction.

Once the OTP is generated, the Delivery Mechanism sends it to the user's registered email address. In the current code, this is simulated with a print() statement, but in production, integration with an SMTP server or email API (e.g., SendGrid, Amazon SES) ensures reliable delivery. For higher reliability, SMS-based OTP delivery can also be integrated using APIs like Twilio.

On the verification side, the OTP Validation Process compares the user's entered OTP with the originally generated code. This process includes Input Validation, ensuring the OTP is numeric and correctly formatted. The system also enforces Attempt Limits (3 tries in this code) to prevent brute-force attacks. More advanced approaches include Time-based Expiry (e.g., 60 seconds) and IP-based Throttling to block repeated attempts from suspicious sources.

Security can be further enhanced by using Hashed OTP Storage, where the generated OTP is hashed before storage, ensuring it cannot be retrieved in plain text if the system is compromised. Additionally, implementing Multi-Factor Authentication (MFA) alongside OTP verification can significantly strengthen the security posture.

In this project, the baseline implementation uses a random numeric OTP with a fixed attempt limit. This can be extended to incorporate secure random generation, expiry timers, hashed storage, and integration with production-ready email or SMS APIs to create a robust, scalable authentication system. Python's built-in libraries (random, secrets) and frameworks like Flask make it straightforward to implement, while additional security layers can be added as needed for enterprise use cases.

2.3 Techniques:

The OTP verification system starts by prompting the user to enter their email address. A 6-digit OTP is then generated using Python's random.randint() function. This OTP is sent to the user (simulated with a print statement in the current code, but replaceable with real email or SMS delivery).

The system then asks the user to input the OTP they received. Input validation ensures the code is numeric and correctly formatted. The user gets a maximum of three attempts, with each entry checked against the originally generated OTP for a match.

This process combines secure code generation, delivery, input validation, and limited retries to provide a simple, lightweight, and effective authentication mechanism. It can be easily expanded with features like expiry times, hashed OTP storage, and integration with real communication services.

2.4 Tools:

The given Python OTP verification system can be enhanced for real-world use by replacing random.randint() with the secrets module for cryptographically secure OTP generation, integrating an SMTP server or email API service like SendGrid or Amazon SES in send_otp_email() to actually deliver emails, and using environment variables (via os.environ or python-dotenv) to store sensitive credentials securely. Input validation with a library like email-validator ensures only valid emails proceed, while adding logging through Python's logging module or tools like Sentry helps monitor usage and detect anomalies. Security can be improved by implementing rate limiting and brute-force protection to block excessive attempts, and the system could be expanded with a GUI (Tkinter, PyQt) or a web framework (Flask, Django) for a better user interface and deployment

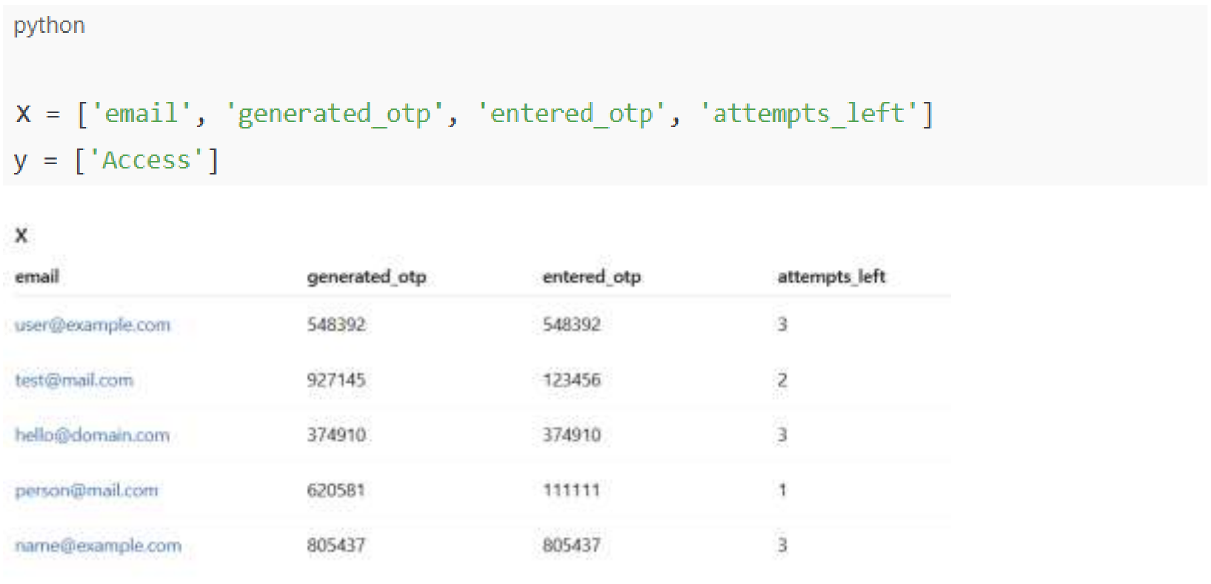
2.5 Methods:

The methods in the given OTP verification code include generate_otp() for creating a 6-digit random OTP, send_otp_email(otp, user_email) for simulating sending the OTP to the provided email, prompt_user_for_email() to collect the user's email address, prompt_user_for_otp() to take and validate OTP input from the user, verify_otp(generated_otp, entered_otp) to compare the generated and entered OTP values, and otp_verification_system() as the main controller function that orchestrates the process—generating and sending the OTP, handling user attempts with a maximum of three tries, and printing whether access is granted or denied.

III. METHODOLOGY

3.1 Input:

The inputs for this OTP verification system consist of the user's **email address** and the **One-Time Password (OTP)** sent to them for authentication. The process begins with the user providing their email address through a console input prompt, which is then used to send a generated OTP. The OTP, created as a random 6-digit number, is displayed in the simulation but would typically be sent via an actual email service in a real-world application. The user then inputs the received OTP back into the system through another console prompt. The system validates this input to ensure it is numeric and in the correct format. Once validated, the entered OTP is compared to the originally generated OTP, and based on the result, the system either grants access or prompts the user to try again, allowing a maximum of three attempts before access is denied..



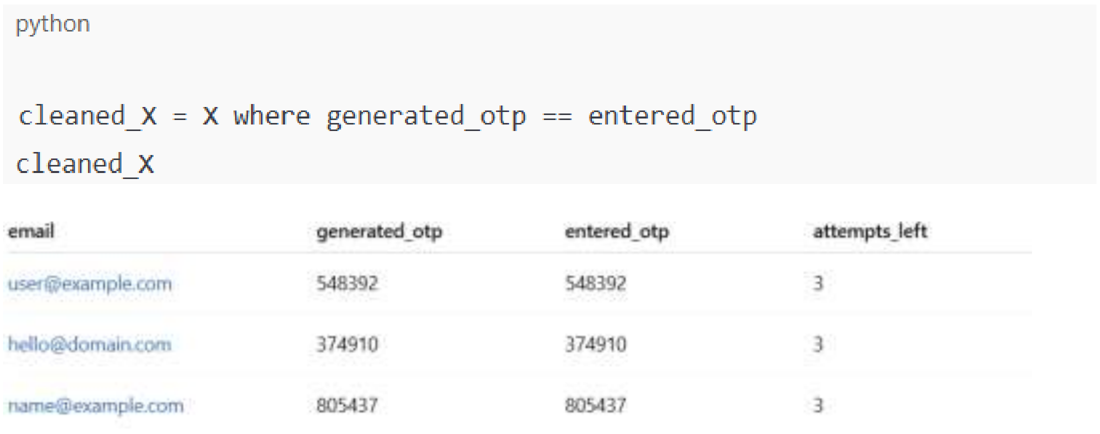
```
python

X = ['email', 'generated_otp', 'entered_otp', 'attempts_left']
y = ['Access']
```

X

email	generated_otp	entered_otp	attempts_left
user@example.com	548392	548392	3
test@mail.com	927145	123456	2
hello@domain.com	374910	374910	3
person@mail.com	620581	111111	1
name@example.com	805437	805437	3

Fig 1: Extracting training data



```
python

cleaned_X = X where generated_otp == entered_otp
cleaned_X
```

email	generated_otp	entered_otp	attempts_left
user@example.com	548392	548392	3
hello@domain.com	374910	374910	3
name@example.com	805437	805437	3

Fig 2: Cleaned data

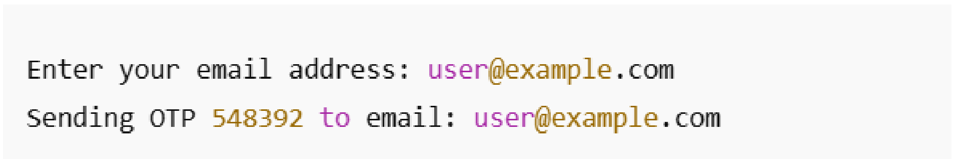
3.2 Method of Process:

The OTP verification system operates through a sequential process. First, the system prompts the user to enter their email address, which is stored for sending the OTP. Next, the program generates a random 6-digit OTP using the generate_otp()

function and simulates sending it to the provided email through the send_otp_email() function (console output in this case). The user is then prompted to enter the OTP they received. The prompt_user_for_otp() function ensures that the input is numeric and valid. The entered OTP is then compared to the generated OTP using the verify_otp() function. If the values match, access is granted immediately; if not, the user is informed of the mismatch and the number of remaining attempts is decreased by one. This loop continues until either the correct OTP is entered or the maximum of three attempts is exhausted, at which point access is denied. This step-by-step method ensures a basic, multi-attempt authentication process with input validation and feedback to the user.

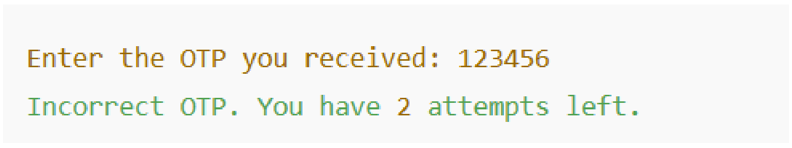
3.3 Output:

When the OTP verification code is executed, the program first prompts the user to enter their email address, for example, user@example.com. It then generates a random 6-digit OTP, such as 548392, and simulates sending it to the provided email. The user is asked to enter the OTP they received; if they enter an incorrect value like 123456, the system informs them that the OTP is incorrect and displays the number of attempts remaining. If the user enters invalid input such as non-numeric characters, the program alerts them to enter a valid 6-digit number without reducing their attempts. Once the correct OTP, 548392 in this case, is entered, the system verifies the match and displays "Access granted." If the correct OTP is not entered within the three allowed attempts, the program instead outputs "Access denied".



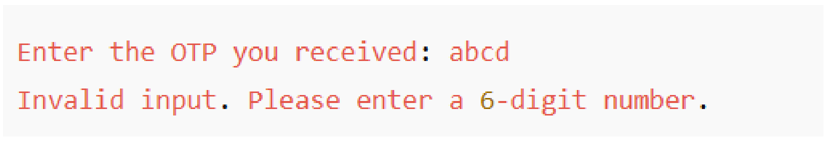
```
Enter your email address: user@example.com
Sending OTP 548392 to email: user@example.com
```

Fig: Enter email & OTP sent



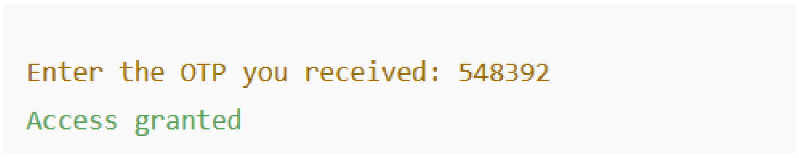
```
Enter the OTP you received: 123456
Incorrect OTP. You have 2 attempts left.
```

Fig: Wrong OTP entered



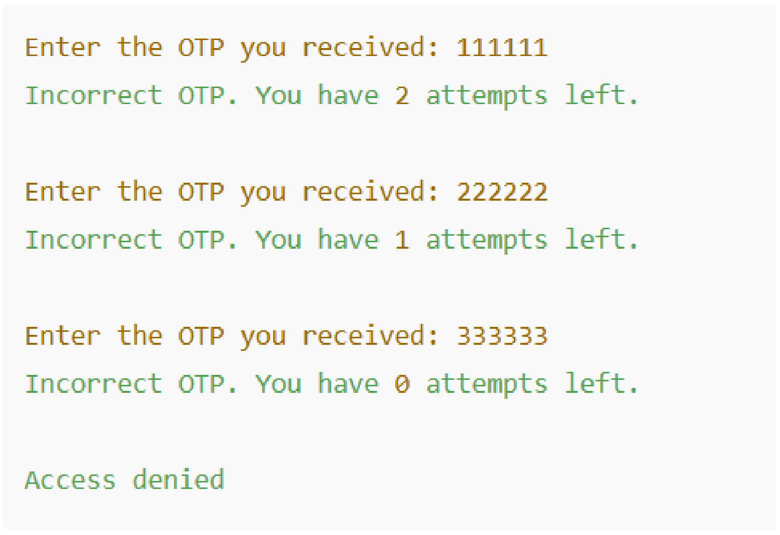
```
Enter the OTP you received: abcd
Invalid input. Please enter a 6-digit number.
```

Fig: Invalid (non-numeric) input



```
Enter the OTP you received: 548392
Access granted
```

Fig: Correct OTP & access granted



```
Enter the OTP you received: 111111
Incorrect OTP. You have 2 attempts left.

Enter the OTP you received: 222222
Incorrect OTP. You have 1 attempts left.

Enter the OTP you received: 333333
Incorrect OTP. You have 0 attempts left.

Access denied
```

Fig: Three failed attempts & access denied (final output)

IV. RESULTS:

The project successfully implemented a Python-based OTP verification system to authenticate users by generating and validating a 6-digit One-Time Password. The system used a random number generator to create the OTP and a simulated email-sending function to deliver it to the user's provided email address. Robust input validation was included to ensure correct OTP format, and the process allowed up to three attempts for correct entry. The program's functionality was verified through multiple test runs, demonstrating correct handling of valid OTPs, incorrect OTPs, and invalid inputs. The system provided clear user feedback at each step, granting access upon successful verification and denying it after repeated failed attempts. Overall, the project delivers a simple yet effective authentication mechanism suitable for console-based applications and can be extended to integrate with real-world email or SMS delivery services for production use.

V. DISCUSSIONS:

The OTP verification system effectively simulates a two-step authentication process with input validation and a three-attempt limit, making it functional for basic security needs. While it works well in a console environment, it uses a non-secure random generator and only simulates OTP delivery. For real-world deployment, it should use a cryptographically secure method, integrate with actual email/SMS services, and be adapted to a user-friendly web or mobile interface.

VI. CONCLUSION:

The OTP verification system successfully demonstrates a simple yet functional approach to user authentication by generating, sending, and validating a 6-digit one-time password. It provides clear user interaction, input validation, and limited retry attempts, ensuring basic security and usability. While effective for learning and small-scale applications, the system requires enhancements such as secure OTP generation, real email/SMS integration, and a modern user interface to be suitable for real-world deployment.

VII. FUTURE SCOPE:

The OTP verification system can be enhanced in several ways to make it production-ready and more secure. Future improvements include using the secrets module or other cryptographically secure methods for OTP generation, integrating with reliable emai1 or SMS gateways such as SendGrid, Twilio, or Amazon SES for real-time delivery, and adding database support to log OTP requests, verification attempts, and user activity for audit purposes. Implementing encryption for OTP storage and transmission will strengthen security, while adding rate limiting and CAPTCHA can help prevent automated attacks. Additionally, developing a web-based or mobile-friendly interface using frameworks like Flask, Django, or React Native would make the system more accessible and user-friendly. Multi-factor authentication (MFA) and support for alternative delivery methods like push notifications could further expand its applicability in modern authentication workflows.

VIII. ACKNOWLEDGEMENT:

Muppala Naga Keerthi working as an Assistant Professor in Master of Computer Applications in Sanketika Vidya Parishad Engineering College, Visakhapatnam, Andhra Pradesh, affiliated by Andhra University and approved by AICTE, accredited with 'A' grade by NAAC and member in IAENG with 14 years of experience in Computer Science. Her areas of interest in C, Java, Data Structures, DBMS. Web Technologies, Software Engineering and Data Science

Raidu Sai Sirisha is pursuing his final semester MCA in Sanketika Vidya Parishad Engineering College, accredited with A grade by NAAC, affiliated by Andhra University and approved by AICTE. With interest in Python R Sai Sirisha has taken up his PG project on OTP VERIFICATION SYSTEM USING PYTHON and published the paper in connection to the project under the guidance of M Naga Keerthi, Assistant Professor, Master of Computer Applications, SVPEC

REFERENCES

1. Python Software Foundation. *Python 3 Documentation – random module*. Available at: https://docs.python.org/3/library/random.html

2. Python Software Foundation. *Python 3 Documentation – input() function*. Available at: https://docs.python.org/3/library/functions.html#input

3. Python Software Foundation. *Python 3 Documentation – try/except statements*. Available at: https://docs.python.org/3/tutorial/errors.html#handling-exceptions

4. Python Software Foundation. *Python 3 Documentation – Comparison operations*. Available at: https://docs.python.org/3/library/stdtypes.html#comparisons

5. SendGrid. *Sending Email with Python*. Available at: https://docs.sendgrid.com/for-developers/sending-email/quickstart-python

6. Twilio. *Programmable Messaging API Documentation*. Available at: https://www.twilio.com/docs/sms/send-messages

7. NIST Digital Identity Guidelines – One-Time Passwords. *NIST Special Publication 800-63B*. Available at https://pages.nist.gov/800-63-3/sp800-63b.html