OTP Verification System

Required Modules:

```
import random
import re
from datetime import timedelta,datetime
```

random: used to generate random values.

re: used to check patterns with help of regular expression.

timedelta: is used to find time difference.

datetime: gives date and time information.

Implement a function to generate a 6-digit OTP randomly.

```
def generate_otp():
  otp = random.randint(100000, 999999)
  return otp
```

Above function is create a random 6 digit number for OTP.

Develop a function to simulate sending the OTP to the user's email address.

```
def send_email(user_email,otp,sender_email="no-reply@example.com"):
  print("generated EMAIL :")
  print("To :",user_email)
  print(" From  :",sender_email)
  print("\n subject : OTP for verification vaild for 3 minutes")
  print("\n your OTP for verification is : ",otp)
```

Above function is to simulate mail to user with OTP.

Where send_email is a function. Parameters user_email is a variable which intakes user's email. Sender_email is default.

Create a function to prompt the user to enter the OTP received in their email.

```
def otp_from_user():
  return input("\n Enter your OTP : ").strip()
```

Above function is to receive OTP from user. Strip is used to remove extra spaces.

Implement a function to verify if the entered OTP matches the generated OTP.

Ensure proper error handling and user-friendly prompts throughout the system.

Allow the user to retry OTP entry in case of incorrect input.

```
def verify_otp(expected_otp, expiry_time, max_attempts=3):
    attempts = 0
    while attempts < max_attempts:
        if datetime.now() > expiry_time:
            print(" OTP has expired. Please request a new one.")
            return False
        user_otp = (otp_from_user()).strip()
        try:
          if not user_otp.isdigit() or len(user_otp) != 6:
            print("\n Enter a valid 6-digit OTP")
            continue
          user_otp=int(user_otp)
        except ValueError:
          print("Enter your valid integer OTP,Letters not allowed")
          continue
        if datetime.now() > expiry_time:
            print("OTP has expired after entry. Please request a new one.")
            return False

        if user_otp == expected_otp:
            print("\n OTP Verified. Access Granted.")
            return True
        else:
            attempts += 1
```

```
            remaining = max_attempts - attempts
            print(f"\n Incorrect OTP. {remaining} attempts remaining.")
    print(" Exceeded maximum attempts. Access Denied.")
    return False
```

Above function is to verify OTP.

- expected_otp: The correct OTP that the user needs to input.
- expiry_time: The datetime when the OTP expires.
- max_attempts: Maximum number of tries allowed to enter the OTP (default is 3).

Initialising variable attempts with 0.

Continues prompting the user until they either succeed or use up all allowed attempts.

If the current time exceeds the expiration time, exit immediately — OTP is no longer valid.

User_otp is input from the user.

Strip removes any leading/trailing spaces.

Checks the input contains only digits and exactly 6 in number.

If valid, converts the string to an integer for comparison.

Handles the case where conversion to integer fails (which shouldn't happen after the .isdigit() check — but added for safety).

Prevents crashing on invalid input like alphabets or special characters.

Double-checks expiration again after the user entered their OTP (helps in edge cases if OTP expires mid-input).

Compares the user's OTP with the expected one.

If they match, access is granted.

If the OTP is incorrect: Increments the attempt counter, displays remaining attempts.

If the loop ends without successful OTP entry: Shows access denied message.

Email pattern validation.

```
def validate_email(user_email):
  pattern = r'^[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,}$'
  if re.match(pattern,user_email):
    return True
  else:
    return False
```

In above function it is used to validate email pattern with help of regular expressions.

- One or more alphanumeric characters, periods, underscores, percentage signs, plus signs, or hyphens before the @ symbol.
- An @ symbol.
- One or more alphanumeric characters, periods, or hyphens after the @ symbol.
- A period followed by two or more alphabetic characters for the top-level domain.

re.match will help in matching the pattern with email pattern. If it matches, then it will pass email.

Displaying timer for expiry.

```
def countdown_timer(expiry_time):
    while True:
        remaining = expiry_time - datetime.now()
        if remaining.total_seconds() <= 0:
            print("\r OTP has expired!\n ")
            break
        mins, secs = divmod(int(remaining.total_seconds()), 60)
        print(f"\r Time remaining: {mins:02d}:{secs:02d}", end="")
        time.sleep(1)
```

Takes one argument: expiry_time, which is a datetime object representing when the OTP will expire.

Continuously calculates the remaining time by subtracting the current time from the expiry time.

If time is up (i.e., remaining seconds are 0 or less), it prints a message and breaks out of the loop.

Converts total seconds into minutes and seconds for a nice format using divmod.

Prints the countdown in MM: SS format.

The \r ensures it prints on the same line, updating every second.

end="" prevents automatic newline after printing.

Pauses the loop for 1 second so the timer updates at real-time pace.

Main code for OTP verification and validation.

```
def otp_verification_system():
    max_regen = 2
    regeneration_count = 0
    email = ""

    while not validate_email(email):
        email = input("Enter your email address: ").strip()
        if not validate_email(email):
            print(" Invalid email format. Please try again.")

    while regeneration_count <= max_regen:
        otp = generate_otp()
        expiry_time = datetime.now() + timedelta(minutes=3)

        send_email(email, otp)
        print(f"\nStarting 3-minute OTP timer... (Regeneration left:
{max_regen - regeneration_count})\n")

        timer_thread = Thread(target=countdown_timer, args=(expiry_time,))
        timer_thread.start()

        verified = verify_otp(otp, expiry_time)
        timer_thread.join()
        if verified:
            break
        else:
            regeneration_count += 1
            if regeneration_count > max_regen:
                print(" You have exceeded the maximum number of OTP
regenerations.")
                print(" Exiting system.")
                break
            retry = input("Would you like to regenerate a new OTP? (y/n):
").strip().lower()
            if retry != 'y':
                print("\n Goodbye.")
                break
            else:
                print(" Generating new OTP...\n")
```

Above code refers following:

max_regen: You allow a maximum of 2 OTP regenerations, so 3 total attempts including the original.

regeneration_count: Keeps track of how many regenerations have been done.

Prompts the user for an email until it matches a valid pattern using validate_email() (presumably a regex-based check).

Generates a new 6-digit OTP and sets its expiry time to 3 minutes from the current time.

Simulates sending OTP (via a send_email() stub).

Starts a countdown timer in a separate thread so the program can still accept user input while the timer runs.

Calls the verify_otp() function (which must return True or False).

Joins the timer thread to ensure it finishes before continuing.

Exits the loop and grants access if the OTP is correct and on time.

If not verified:

Increments regeneration counter.

Checks if max regenerations are exceeded.

Asks the user if they want to regenerate or exit.

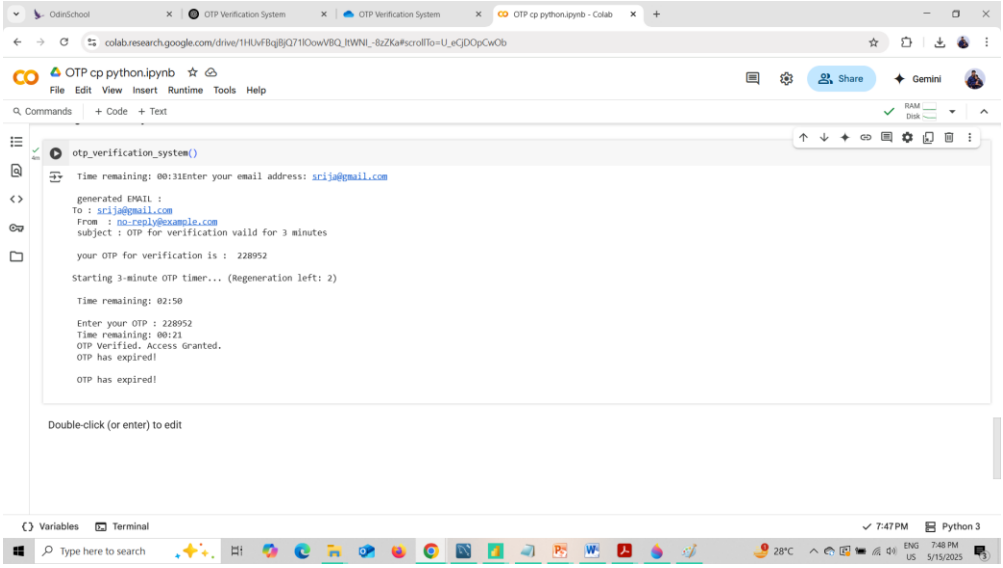
Process to run the program, and any dependencies required.

- OTP verification system does not require any external or third-party libraries. It uses only built-in modules.
- And make sure using python Python 3.x (Recommended: Python 3.6 or later).
- Save the script in a file.
- Make sure all the dependences are installed.
- Include all functions: generate_otp, send_email, validate_email, otp_from_user, verify_otp,countdown_timer and otp_verification_system.
- Run the script.

TEST CASES:

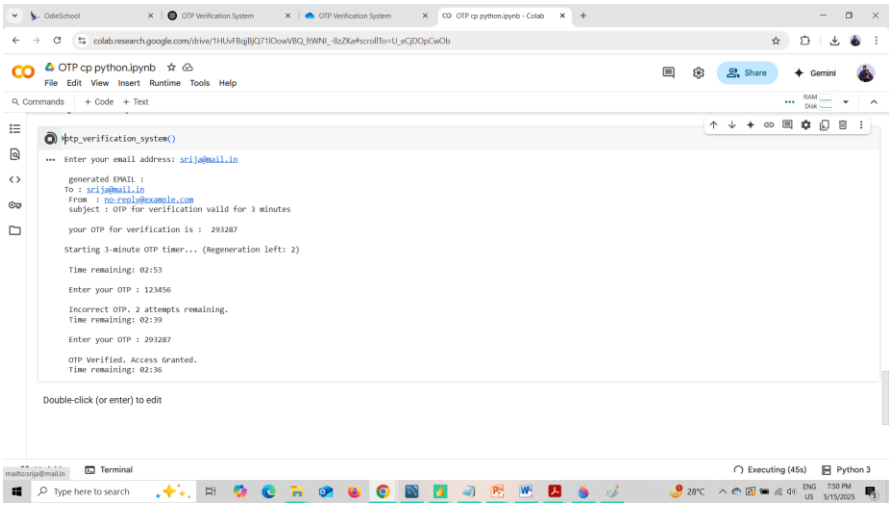
Valid Email and Correct OTP on First Try

- Valid email
- Valid OTP in 1st try.



Valid Email, Incorrect OTP Then Correct OTP

- Valid email.
- First OTP entry: incorrect
- Second OTP entry: correct
- Within expiry time

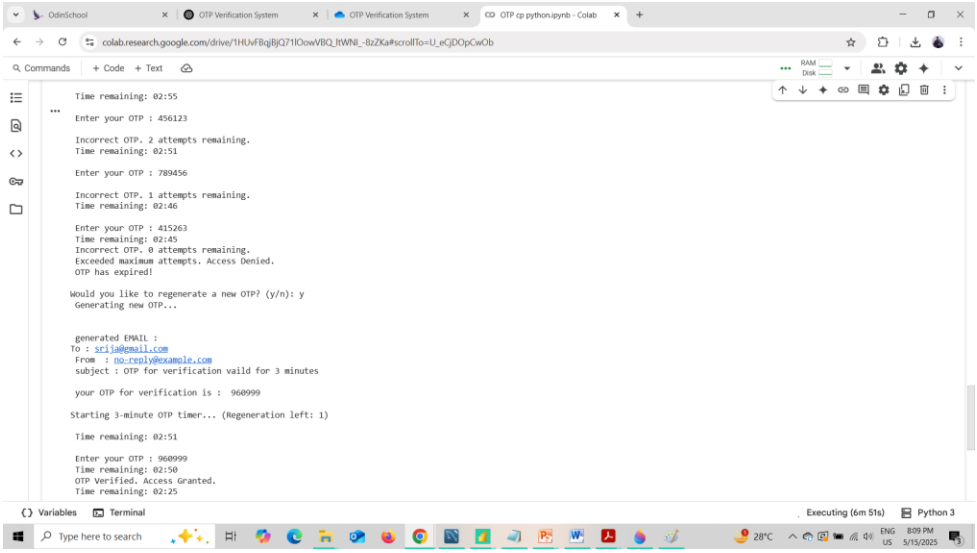


OTP Expired:

- Valid email.
- Correct OTP after OTP expiry.

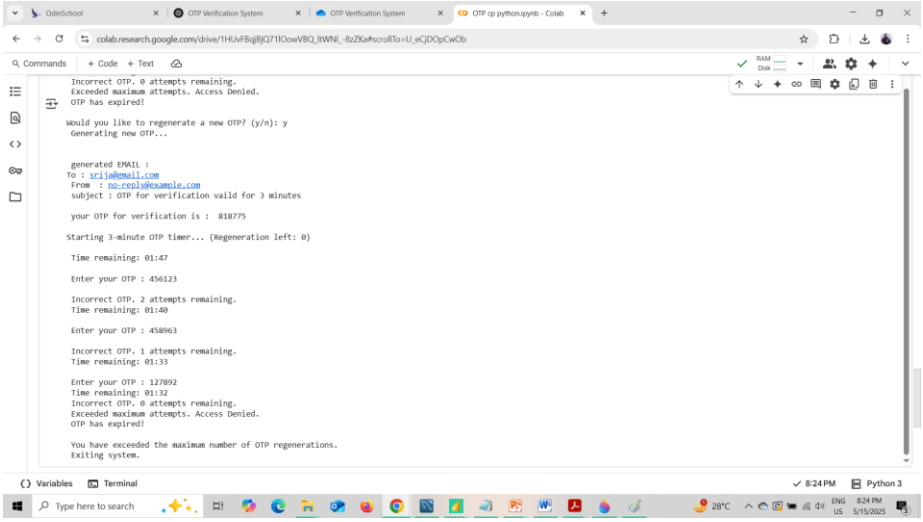
OTP Regeneration (Within Limit):

- Valid email.
- 3 attempts failure within expiry time.
- Asks for regeneration.



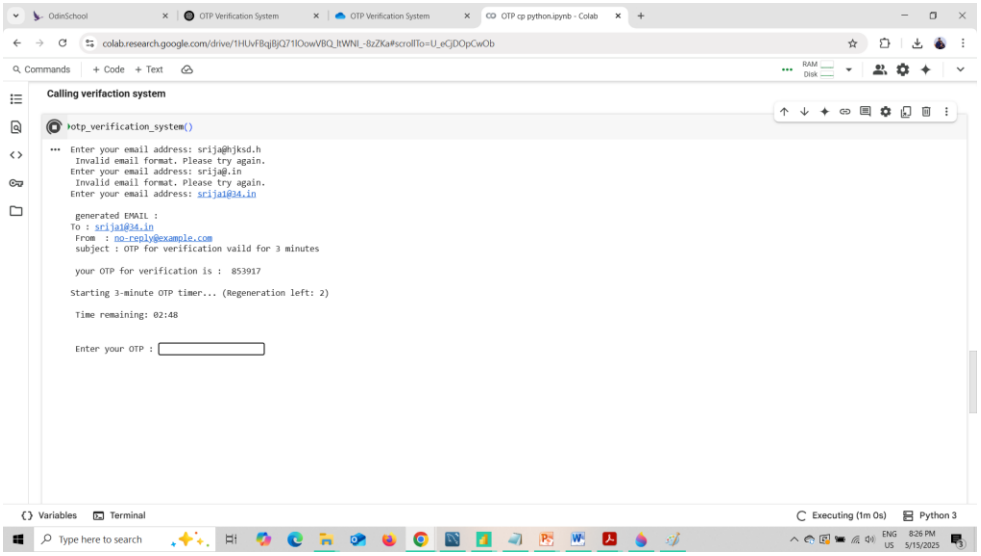
OTP Regeneration Limit Exceeded:

- Valid email.
- Regeneration limit exceeded. (here its 3).



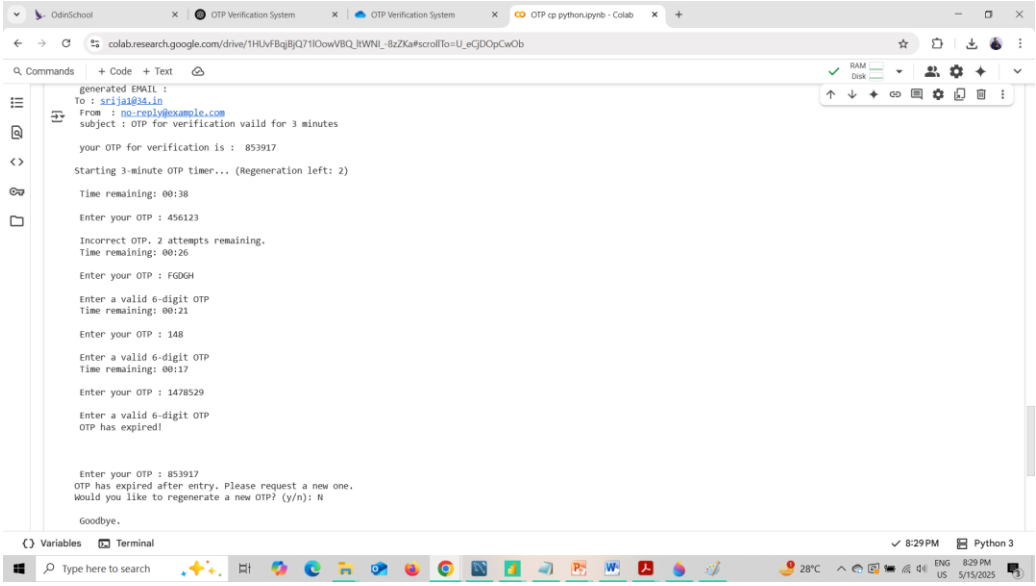
Invalid Email Format:

- Entering an invalid email.
- Asks for valid email to enter.



OTP with Non-Numeric or Short Input:

- Valid email.
- Enter OTP with string and non-6-digit OTP.



User Declines OTP Regeneration:

- Valid email.
- Entering wrong OTP for 3 times.
- Declining for regenration as ‘N’ or ‘n’.