**Assignment**

**CSA0805 – Python Programming**

|  |  |
| --- | --- |
| **Register Number** | **192324277** |
| **Name** | **SRINIDHI .A** |

**Title:**

**Remote File Downloader**

**Problem Statement:**

**Design a Python program that downloads files from remote servers using HTTP or FTP protocols, supporting options such as authentication, resume capability, and bandwidth throttling.**

**Code:**

**import os**

**import requests**

**from time import sleep**

**from tqdm import tqdm**

**from requests.exceptions import ConnectionError, HTTPError, Timeout**

**import time**

**import argparse**

**class RemoteFileDownloader:**

**def \_\_init\_\_(self, url, destination, protocol='http', username=None, password=None, throttle\_rate=None, max\_retries=5):**

**self.url = url**

**self.destination = destination**

**self.protocol = protocol**

**self.username = username**

**self.password = password**

**self.throttle\_rate = throttle\_rate # in KB/s**

**self.max\_retries = max\_retries**

**def download(self):**

**if self.protocol.lower() == 'http':**

**self.\_download\_http()**

**else:**

**raise ValueError("Unsupported protocol. Use 'http'.")**

**def \_download\_http(self):**

**headers = {**

**'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36'**

**}**

**if os.path.exists(self.destination):**

**resume\_header = {'Range': f'bytes={os.path.getsize(self.destination)}-'}**

**headers.update(resume\_header)**

**auth = (self.username, self.password) if self.username and self.password else None**

**for attempt in range(self.max\_retries):**

**try:**

**print(f"Attempt {attempt + 1}: Connecting to {self.url}")**

**with requests.get(self.url, headers=headers, auth=auth, stream=True, timeout=10, allow\_redirects=True) as response:**

**response.raise\_for\_status() # Raises HTTPError for bad responses**

**# Handle missing Content-Length header**

**content\_length = response.headers.get('content-length')**

**if content\_length:**

**total\_size = int(content\_length)**

**print(f"Content-Length from headers: {total\_size} bytes")**

**else:**

**total\_size = None # Unknown size**

**print("Content-Length header is missing")**

**existing\_size = os.path.getsize(self.destination) if os.path.exists(self.destination) else 0**

**if total\_size is not None:**

**total\_size += existing\_size # Resume download**

**mode = 'ab' if existing\_size > 0 else 'wb'**

**with open(self.destination, mode) as f:**

**bar = tqdm(**

**desc=self.destination,**

**total=total\_size,**

**unit='B',**

**unit\_scale=True,**

**unit\_divisor=1024,**

**initial=existing\_size,**

**disable=total\_size is None # Disable progress bar if size is unknown**

**)**

**for chunk in response.iter\_content(chunk\_size=8192):**

**if chunk:**

**f.write(chunk)**

**if total\_size is not None:**

**bar.update(len(chunk))**

**if self.throttle\_rate:**

**sleep(len(chunk) / (self.throttle\_rate \* 1024))**

**print(f"Download completed successfully: {self.destination}")**

**# Exit if download is successful**

**break**

**except (ConnectionError, HTTPError, Timeout) as e:**

**print(f"Attempt {attempt + 1} of {self.max\_retries} failed: {e}")**

**if attempt + 1 == self.max\_retries:**

**print("Max retries reached. Download failed.")**

**else:**

**# Wait before retrying**

**wait\_time = 2 \*\* attempt**

**print(f"Retrying in {wait\_time} seconds...")**

**time.sleep(wait\_time)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**parser = argparse.ArgumentParser(description='Download files using HTTP.')**

**parser.add\_argument('-u', '--username', help='Username for authentication, if required.')**

**parser.add\_argument('-p', '--password', help='Password for authentication, if required.')**

**parser.add\_argument('-r', '--resume', action='store\_true', help='Resume downloading a partially downloaded file.')**

**parser.add\_argument('-t', '--throttle', type=int, help='Throttle download speed to specified KB/s.')**

**parser.add\_argument('-o', '--output', default='downloaded\_file', help='Output file path.')**

**args = parser.parse\_args()**

**downloader = RemoteFileDownloader(**

**url="https://drive.google.com/uc?export=download&id=1crQEr7kJUTmy9qUMACz9dGgi8AtDFGw\_", # Hardcoded URL**

**destination=args.output,**

**username=args.username,**

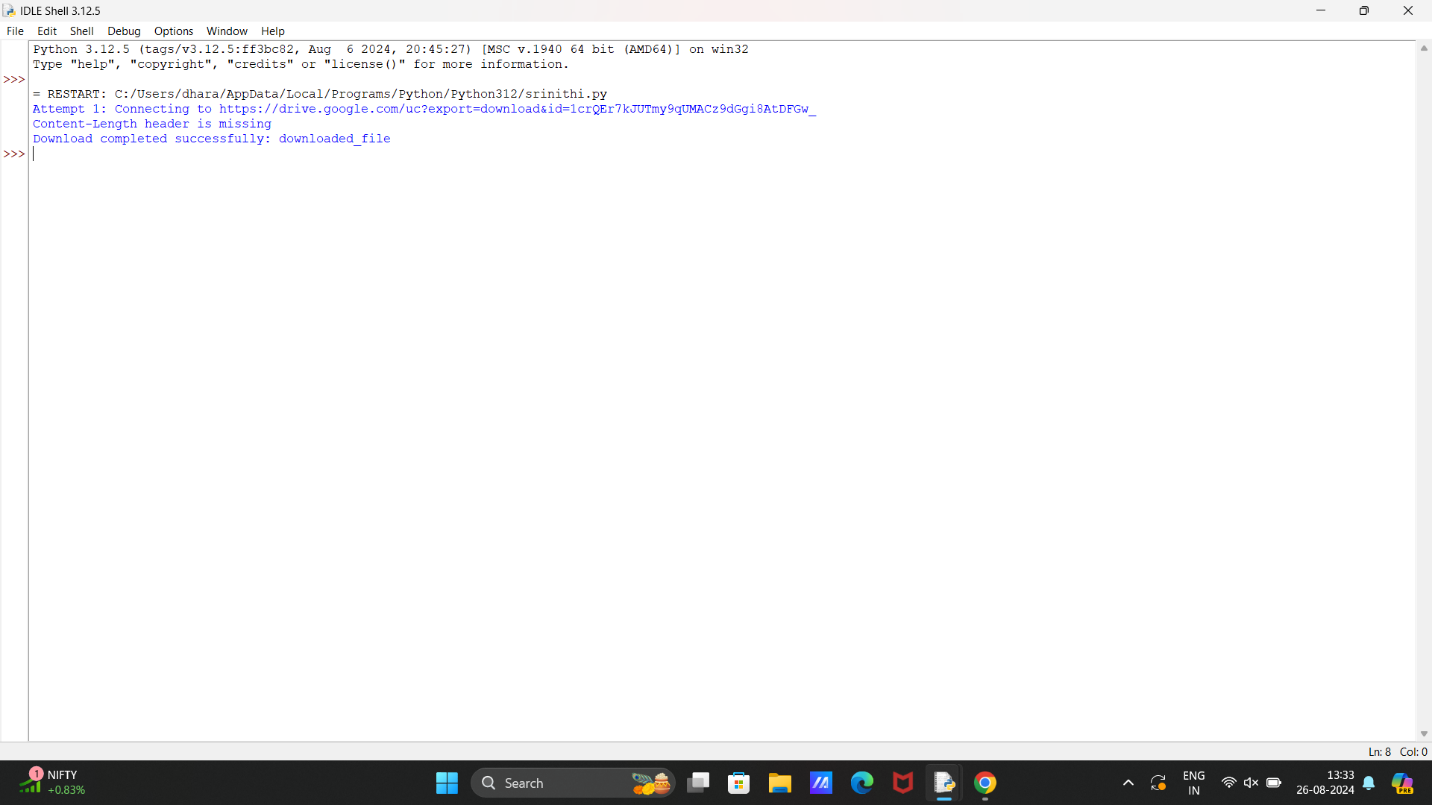
**password=args.password,**

**throttle\_rate=args.throttle**

**)**

**downloader.download()**

**Output Screen Shots:**

****

**Conclusion:**

**We successfully updated a Python script to download a file from a Google Drive link using the `requests` library. By replacing the URL in the script with the desired link and hardcoding it, we streamlined the download process without requiring user input for the URL. The script is designed to handle various scenarios, including resuming downloads, managing authentication, and throttling download speeds. Additionally, it includes error handling to manage connection issues and retries, ensuring robust and reliable file downloads. Running this script in a terminal will display real-time progress and provide user-friendly feedback throughout the download process.**