**Python Image Scraper with Folder Organization**

Purpose:

This Python script is designed to scrape images from Google search based on a user-provided query and save these images into a separate folder named after the search query.

Libraries Used:

* **os**: Used to create directories for organizing images.
* **requests**: Used to send HTTP GET requests and download images.
* **time**: Used for adding delays to ensure the page loads properly.
* **BeautifulSoup**: Used for parsing HTML content.
* **selenium**: Used to automate the web browser (Chrome) for scrolling and interaction with the webpage.
* **re**: Used for regular expressions.

Functions and Explanation:

1. **scroll\_down(driver, num\_scrolls)**: This function scrolls down the webpage to load more images. It takes a Selenium WebDriver (**driver**) and the number of times to scroll (**num\_scrolls**) as arguments.
2. **scrape\_images(query, num\_images)**: This is the main function for scraping images. It takes two parameters: the search query (**query**) and the number of images to scrape (**num\_images**).
   * It creates a directory called "images" if it doesn't already exist. This directory will contain subfolders for different search queries.
   * It sets up the Selenium WebDriver for Google Chrome and navigates to the Google Images search results page for the provided query.
   * It initializes a counter for downloaded images.
   * The script then enters a loop to scroll down the page (**num\_scrolls** times) and load more images.
   * Inside the loop, it parses the HTML content of the search page using BeautifulSoup.
   * It finds all image tags in the HTML and iterates through them to download and save each image.
   * For each image, it checks if the image URL is a valid HTTP or HTTPS link and starts with "http" or "https". If so, it proceeds to download the image.
   * It constructs a unique image file name based on the search query and image count and saves the image to a subfolder named after the search query.
   * The script handles both regular image URLs and base64-encoded images (optional).
   * The loop continues until the desired number of images (**num\_images**) have been downloaded.
   * The script handles exceptions and errors gracefully, displaying error messages if any occur.
3. Input: The user is prompted to enter a search query and the number of images to scrape.
4. The **scrape\_images** function is called with the user's input to start the scraping process.

Output:

* The script will download the specified number of images from the Google Images search results for the given query.
* All downloaded images will be organized into subfolders within the "images" directory, with each subfolder named after the search query.
* The script will display messages indicating the progress and status of the scraping process.

By following this structure, you can easily customize the code for different scraping tasks and organize the downloaded images based on your search queries.



create a documention for the code

ChatGPT

Sure, here's documentation for the Python image scraper code that scrapes images from Google search and saves them to separate folders based on the search query:

**Python Image Scraper with Folder Organization Documentation**

**Overview**

This Python script is designed to scrape images from Google search based on a user-provided query and save these images into separate folders named after the search query. The code uses libraries such as **requests**, **BeautifulSoup**, **selenium**, and **os** for web scraping and file organization.

**Table of Contents**

1. [Dependencies](https://chat.openai.com/c/c98ef2f4-5ed1-4487-bcde-1bcf621a70a7#dependencies)
2. [Functions](https://chat.openai.com/c/c98ef2f4-5ed1-4487-bcde-1bcf621a70a7#functions)
   * [scroll\_down](https://chat.openai.com/c/c98ef2f4-5ed1-4487-bcde-1bcf621a70a7#scroll_down)
   * [scrape\_images](https://chat.openai.com/c/c98ef2f4-5ed1-4487-bcde-1bcf621a70a7#scrape_images)
3. [Usage](https://chat.openai.com/c/c98ef2f4-5ed1-4487-bcde-1bcf621a70a7#usage)
4. [Output](https://chat.openai.com/c/c98ef2f4-5ed1-4487-bcde-1bcf621a70a7#output)
5. [Customization](https://chat.openai.com/c/c98ef2f4-5ed1-4487-bcde-1bcf621a70a7#customization)

**Dependencies<a name="dependencies"></a>**

This code relies on several Python libraries:

* **os**: Used for creating directories and managing file paths.
* **requests**: Enables sending HTTP GET requests and downloading images.
* **time**: Provides functionality to add delays for proper page loading.
* **BeautifulSoup**: Used for parsing HTML content.
* **selenium**: Automates web browser interactions, scrolling, and navigation.
* **re**: Provides regular expression support (optional).

Before running the script, ensure that these libraries are installed in your Python environment. You can install missing libraries using the following commands:

bashCopy code

pip install requests pip install beautifulsoup4 pip install selenium

Additionally, you need to have the [Chrome WebDriver](https://sites.google.com/chromium.org/driver/) installed and its path configured in the system's PATH variable for Selenium to work.

**Functions<a name="functions"></a>**

**scroll\_down(driver, num\_scrolls)<a name="scroll\_down"></a>**

This function scrolls down a webpage to load more images. It takes two parameters:

* **driver**: A Selenium WebDriver instance.
* **num\_scrolls**: The number of times to scroll down the webpage.

**scrape\_images(query, num\_images)<a name="scrape\_images"></a>**

This is the main function for scraping images. It takes two parameters:

* **query**: The search query for Google Images.
* **num\_images**: The number of images to scrape.

The function performs the following steps:

1. Creates a directory called "images" if it doesn't already exist. This directory will contain subfolders for different search queries.
2. Sets up the Selenium WebDriver for Google Chrome and navigates to the Google Images search results page for the provided query.
3. Initializes a counter for downloaded images.
4. Enters a loop to scroll down the page (**num\_scrolls** times) and load more images.
5. Parses the HTML content of the search page using BeautifulSoup.
6. Finds all image tags in the HTML and iterates through them to download and save each image.
7. Constructs a unique image file name based on the search query and image count and saves the image to a subfolder named after the search query.
8. Handles both regular image URLs and base64-encoded images (optional).
9. Continues the loop until the desired number of images (**num\_images**) have been downloaded.
10. Handles exceptions and errors gracefully, displaying error messages if any occur.

**Usage<a name="usage"></a>**

1. Run the script.
2. Enter the search query when prompted.
3. Specify the number of images to scrape.
4. The script will proceed to scrape images and organize them into folders.

**Output<a name="output"></a>**

The script will download the specified number of images from the Google Images search results for the given query. All downloaded images will be organized into subfolders within the "images" directory, with each subfolder named after the search query. Progress and status messages will be displayed during the scraping process.

**Customization<a name="customization"></a>**

You can customize the script for different scraping tasks and modify it as needed:

* Adjust the number of scrolls to load more images by changing the **num\_scrolls** variable.
* Customize the file naming convention by modifying the **img\_name** variable.
* Extend the script to handle additional types of image URLs or formats if required.
* Implement error handling and reporting specific to your use case.

This documentation provides an overview of the Python image scraper code and explains its functions, usage, output, and customization options. You can adapt and extend this code to suit your specific web scraping needs while maintaining organized storage of scraped images.