**Amazon Web Scraper Project**

# importing libraries

**from** bs4 **import** BeautifulSoup

**import** requests

**import** smtplib

**import** time

**import** datetime

# **from** bs4 **import** BeautifulSoup

Beautiful Soup is a popular Python library used for web scraping and parsing HTML and XML documents. It provides a convenient way to extract data from web pages by navigating the parse tree, searching for specific elements, and manipulating the HTML or XML structure.

With Beautiful Soup, you can easily extract data from HTML documents and XML files by using intuitive methods and CSS selectors. It handles a variety of parsing tasks, such as converting incoming documents to Unicode, automatically closing non-closed tags, and transforming malformed markup into well-formed HTML or XML.

By including this import statement at the beginning of your Python script, you gain access to the functionality provided by Beautiful Soup, such as parsing HTML or XML, searching for elements, and extracting data from web pages.

# **import** requests

In Python, the **import requests** statement is used to import the **requests** module into your code. The **requests** module is a popular Python library that allows you to send HTTP requests and interact with web services.

Once you import the **requests** module, you can use its functions and classes to perform various HTTP operations such as making GET and POST requests, handling cookies, managing sessions, and more.

# **import** smtplib

In Python, the **import smtplib** statement is used to import the **smtplib** module, which provides an interface for sending emails using the Simple Mail Transfer Protocol (SMTP).

Once you import the **smtplib** module, you can use its classes and functions to establish an SMTP connection to an email server and send emails.

**# import** time

In Python, the **import time** statement is used to import the **time** module, which provides various functions related to time measurement and manipulation.

Once you import the **time** module, you can use its functions to perform actions such as pausing the program execution, measuring the current time, converting between different time representations, and more.

**# import** datetime

In Python, the **import datetime** statement is used to import the **datetime** module, which provides classes and functions for working with dates and times.

Once you import the **datetime** module, you can use its classes like **datetime**, **date**, **time**, **timedelta**, and functions to perform various operations involving dates, times, and timedeltas.

User-Agent

In Python, the User-Agent refers to the string that is included in the HTTP request headers sent by a client application when making a request to a server. It provides information about the client software, such as the browser, operating system, and other relevant details.

To set the User-Agent string in Python when making HTTP requests using libraries like **requests**, you can modify the headers of the request. Here's an example:

import requests

url = 'https://www.example.com'

headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36'}

response = requests.get(url, headers=headers)

#writing Data inside csv file

with open('Amazon\_Web\_Scraper\_Dataset.csv', 'w', newline='', encoding = 'UTF8') as f:

writer = csv.writer(f)

writer.writerow(header)

writer.writerow(data)

In this code, the **with** statement is used to open the CSV file in write mode, ensuring that it is properly closed after writing. The **csv.writer** object is created, specifying the file object **f** as the output destination.

The **writer.writerow(header)** writes the header row to the CSV file, and **writer.writerow(data)** writes the data row to the file.

In this example, the code opens the file named "Amazon\_Web\_Scraper\_Dataset.csv" in write mode using the **open()** function. The **with** statement ensures that the file is properly closed after writing.

The **csv.writer** object is created with the file object **f**, and the **writer.writerow(header)** writes the header row to the CSV file. Then, **writer.writerow(data)** writes the data row to the file.

You can modify the **header** and **data** variables to contain the actual header and data you want to write to the CSV file. Each **writer.writerow()** call writes a single row to the file.

Ensure that you have the necessary permissions to write to the specified file location.

The **encoding='UTF8'** parameter in the **open()** function specifies the encoding to be used when writing to the CSV file. UTF-8 is a widely-used character encoding that can represent any character in the Unicode standard, making it suitable for handling various languages and special characters.

When writing to a CSV file, it is important to ensure that the encoding matches the expected encoding for the data you are working with. UTF-8 is a common choice as it supports a wide range of characters.

TF-8 (Unicode Transformation Format-8) is a character encoding that is widely used for representing Unicode characters.

The ‘**with’** statement in Python provides a convenient way to manage resources and handle exceptions in a clean and concise manner. It ensures that a context is properly entered and exited, allowing for automatic cleanup and resource management.

Both methods for reading csv are correct :

data1= pd.read\_csv('C:/Users/Chetan/Amazon\_Web\_Scraper\_Dataset.csv')

data1

df = pd.read\_csv(r'C:\Users\Chetan\Amazon\_Web\_Scraper\_Dataset.csv')

df

**# Now we are appending data to the csv**

with open('Amazon\_Web\_Scraper\_Dataset.csv', 'a+', newline='', encoding = 'UTF8') as f:

writer = csv.writer(f)

writer.writerow(data)

# every time its difficult to update manually, so we need autoappending each time

**# auto appendinding**

def check\_price():

URL = "https://www.amazon.in/Puma-Solid-Regular-T-Shirt-58666927\_Desert/dp/B0BRN2JWWV/ref=sr\_1\_16?crid=CRZMQJETCO5E&keywords=puma+tshirt&qid=1685606359&sprefix=puma+tshirt%2Caps%2C270&sr=8-16"

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"}

page = requests.get(URL, headers=headers)

soup1 = BeautifulSoup(page.content, "html.parser")

soup2 = BeautifulSoup(soup1.prettify(), "html.parser")

title = soup2.find(id = 'productTitle').get\_text()

price = soup2.find(class\_='a-offscreen').get\_text()

title = title.strip()

price = price.strip()[1:]

import datetime

today = datetime.date.today()

import csv

header = ['Title', 'Price', 'Date']

data = [title, price, today]

with open('Amazon\_Web\_Scraper\_Dataset.csv', 'a+', newline='', encoding = 'UTF8') as f:

writer = csv.writer(f)

writer.writerow(data)

#we created function for auto append where all process are there

import datetime

today = datetime.date.today() # it will give today date

**# putting timer**

So that every selected time it will need to append

while(True): # while loop is True then go for check\_price() function

check\_price()

time.sleep(86400)

#for every 86400's(1 day) it will run this entire process, check\_price()

**Seeing data**

df= pd.read\_csv('C:/Users/Chetan/Amazon\_Web\_Scraper\_Dataset.csv')

print(df)

case study:

we want to buy a tshirt which values is 529 rupees, but we wanted that tshirt to be come with 425 rupees, during sales time, there will drop change happens, when tshirt price below 425, that time automatically Email have to come saying that, “tshirt price is low, best time to buy”

solution: as we set append data automatically for every 86400 seconds that means every each day,

if any day , price becomes less than 425 , it will need to send mail

if(price<425):

send\_mail()

this condition needs to put in auto appending code (function)i.e,.

def check\_price():

URL = "https://www.amazon.in/Puma-Solid-Regular-T-Shirt-58666927\_Desert/dp/B0BRN2JWWV/ref=sr\_1\_16?crid=CRZMQJETCO5E&keywords=puma+tshirt&qid=1685606359&sprefix=puma+tshirt%2Caps%2C270&sr=8-16"

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"}

page = requests.get(URL, headers=headers)

soup1 = BeautifulSoup(page.content, "html.parser")

soup2 = BeautifulSoup(soup1.prettify(), "html.parser")

title = soup2.find(id = 'productTitle').get\_text()

price = soup2.find(class\_='a-offscreen').get\_text()

title = title.strip()

price = price.strip()[1:]

import datetime

today = datetime.date.today()

import csv

header = ['Title', 'Price', 'Date']

data = [title, price, today]

with open('Amazon\_Web\_Scraper\_Dataset.csv', 'a+', newline='', encoding = 'UTF8') as f:

writer = csv.writer(f)

writer.writerow(data)

if(price < 425):

send\_mail() # if price<425, it need to go send\_mail() function

# If you want to try sending yourself an email (just for fun) when a price hits

# below certain level you can try it out with this script

def send\_mail():

server = smtplib.SMTP\_SSL('smtp.gmail.com',465)

server.ehlo()

#server.startls()

server.ehlo()

server.login('chetan.patil260598@gmail.com','123cap999')

subject = "The Shirt you want is below $15! Now is your chance to buy!"

body = "Chetan, This is the moment we have been waiting for. Now is your chance to pick up the shirt of your dreams. Don't mess it up! Link here:

https://www.amazon.in/Puma- Solid-Regular-T-Shirt-58666927\_Desert /dp/B0BRN2JWWV/ref=sr\_1\_16?crid= CRZMQJETCO5E&keywords=puma+tshirt&qid=1685606359&sprefix=puma+tshirt%2Caps%2C270&sr=8-16"

msg = f"Subject: {subject}\n\n{body}"

server.sendmail(

'chetan.patil260598@gmail.com',

msg

)

print("Hey, Email Has Been Sent!!!")

**Explanation:**

Here is a breakdown of the code:

* The first line imports the smtplib module, which provides a Python interface to the SMTP protocol.
* The second line creates a new SMTP\_SSL object, which represents an SMTP server that uses the Secure Sockets Layer (SSL) protocol to encrypt traffic.
* The third line calls the ehlo() method on the SMTP\_SSL object to identify the client to the server.
* The fourth line (which is commented out) calls the starttls() method on the SMTP\_SSL object to start a secure connection.
* The fifth line calls the ehlo() method again to re-identify the client to the server.
* The sixth line calls the login() method on the SMTP\_SSL object to authenticate the client with the server.
* The seventh line defines the subject and body of the email.
* The eighth line constructs the message body by combining the subject and body strings.
* The ninth line calls the sendmail() method on the SMTP\_SSL object to send the email.

In this updated code:

1. The **smtplib** module is imported at the beginning.
2. The **server.sendmail()** function is modified to include the recipient's email address as the second parameter. Replace **'recipient@example.com'** with the actual email address of the recipient.
3. The **server.quit()** method is called after sending the email to gracefully close the SMTP connection.

Make sure to replace **'xxxxxxxxxxxxxx'** with the correct password for the sender's email account.

Remember to enable "less secure apps" access or use an app-specific password if you encounter any authentication issues with your Gmail account.

The line **server = smtplib.SMTP\_SSL('smtp.gmail.com', 465)** sets up an SMTP connection to the Gmail server using the SSL (Secure Sockets Layer) protocol on port 465.

Here's a breakdown of the code:

* **smtplib** is the Python library for sending emails using the SMTP protocol.
* **SMTP\_SSL()** is a method provided by the **smtplib** library to establish an SMTP connection over SSL.
* **'smtp.gmail.com'** is the SMTP server address for Gmail.
* **465** is the default SSL/TLS port for Gmail's SMTP server.

By calling **smtplib.SMTP\_SSL('smtp.gmail.com', 465)**, you create an SMTP object (**server**) that is ready to communicate with Gmail's SMTP server securely using SSL encryption.

After creating the **server** object, you can further configure it and perform operations such as authentication, sending emails, and closing the connection.